

```

// AddDlg.h
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#ifndef AddDlgH
#define AddDlgH
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//-----
class TForm2 : public TForm
{
__published: // IDE-managed Components
    TButton *Button1;
    TButton *Button2;
    TLabel *Label1;
    TComboBox *CBDataType;
    TLabel *LblRegEx;
    TEdit *EdData;
    TLabel *Label3;
    TEdit *EdName;
    TLabel *Label2;
    void __fastcall CBDataTypeChange(TObject *Sender);
private: // User declarations
    AnsiString __fastcall GetCurRegEx(void);
    AnsiString __fastcall GetCurFormat(void);
public: // User declarations
    __fastcall TForm2(TComponent* Owner);
};
//-----
void AddItemToLockBox(void);
//-----
#endif
// LbStore.h
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#ifndef LbStoreH
#define LbStoreH
#include "LockPub.h"
//-----
bool InternalAddItemToLockbox(TLockboxItem* lbi);
bool InternalRemoveItemFromLockbox(DWORD dwItemID);
bool InternalGetLockboxItems(PLockboxItem pli, DWORD* dwItemCnt);
#endif
// LockPriv.h
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#ifndef LockPrivH
#define LockPrivH

```

```

#include <boost\regex.hpp>
//-----
enum TRegExChars {REC_NONE, REC_ALPHA, REC_UPPER, REC_LOWER, REC_DIGIT, REC_SPACE,
REC_OTHER};
typedef boost::match_results<std::string::const_iterator> regexp_match_results;
// EscapeRegExString() forms a regular expression from a LBDT_STRING type
// lockbox entry by escape reserved regex characters
std::string BuildRegExString(const std::string str, bool bCaseSensitive);
// MD5Hash() returns a base64 encoded MD5 hash of the provided buffer
std::string MD5Hash(unsigned char *buf, unsigned buflen);
// FindIt() searches the szStr buffer for the szExpression regular expression,
// and, if found, returns the found data formatted with szFormat
std::string FindIt(const char* szStr, const char* szExpression, const char* szFormat);
bool IsTextInLockbox(char*szStr);
#endif
// LockPub.h
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#ifndef LockPubH
#define LockPubH
#include <windows.h>
//-----
enum TLockboxDataType {LBDT_STRING, LBDT_STRING_CI, LBDT_USPHONE, LBDT_SSN, LBDT_VISAMC,
LBDT_AMEX};
extern const char* g_StandardExpressions[];
extern const char* g_StandardFormats[];
struct TLockboxItem
{
    DWORD dwItemID;
    TLockboxDataType lbdt;
    char szDescription[128];
    char szHash[32];
    char szRegEx[256];    // used for LBDT_STRING & LBDT_STRING_CI
};
typedef TLockboxItem* PLockboxItem;
BOOL WINAPI tvAddItemToLockbox(TLockboxDataType lbdt, char* szData,
    char* szDescription, DWORD* dwItemID);
BOOL WINAPI tvRemoveItemFromLockbox(DWORD dwItemID);
BOOL WINAPI tvGetLockboxItems(PLockboxItem pli, DWORD* dwItemCnt);
BOOL WINAPI tvSetLockboxItems(PLockboxItem pli, DWORD dwItemCnt);
#endif
// Main.h
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#ifndef MainH
#define MainH
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>

```

```

#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ActnList.hpp>
#include <ActnMan.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <ToolWin.hpp>
//-----
class TForm1 : public TForm
{
__published: // IDE-managed Components
    TGroupBox *GroupBox1;
    TGroupBox *GroupBox2;
    TMemo *MemSample;
    TButton *BtnTest;
    TButton *BtnExit;
    TListView *LVLockbox;
    TPopupMenu *PopupMenu1;
    TActionManager *ActionManager1;
    TAction *ActAddItem;
    TAction *ActRemoveItem;
    TMenuItem *MIAdd;
    TMenuItem *MIRemove;
    TAction *ActLoadFile;
    TPopupMenu *PopupMenu2;
    TMenuItem *Loadfile1;
    TOpenDialog *OpenDialog;
    TToolBar *ToolBar1;
    TToolButton *ToolButton1;
    TToolButton *ToolButton2;
    TToolButton *ToolButton3;
    TToolButton *ToolButton4;
    void __fastcall BtnTestClick(TObject *Sender);
    void __fastcall BtnExitClick(TObject *Sender);
    void __fastcall ActRemoveItemUpdate(TObject *Sender);
    void __fastcall ActAddItemExecute(TObject *Sender);
    void __fastcall ActRemoveItemExecute(TObject *Sender);
    void __fastcall ActLoadFileExecute(TObject *Sender);
    void __fastcall LVLockboxKeyUp(TObject *Sender, WORD &Key,
        TShiftState Shift);
private: // User declarations
    void RefreshLockboxView(void);
public: // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif
// base64_enc.h

```

```

#ifndef __BASE64_ENC_H__
#define __BASE64_ENC_H__
// existing stream classes like faststream don't like you mucking around with index values.
// emulate a macintosh handle mem object
#define MH_SIZE 2048 // this should be plenty big for an url
class MacHandle
{
public:
    MacHandle()
    {
        Initialize(0);
    }
    MacHandle(UINT size)
    {
        Initialize(size);
    }
    MacHandle( BYTE* pszData, UINT length )
    {
        if ( Initialize(length) )
        {
            SetHandleData( pszData, length );
        }
    }
    ~MacHandle()
    {
        if ( m_pData ) delete [] m_pData;
    }
    bool Initialize(UINT newsize)
    {
        m_pData = new BYTE[MH_SIZE];
        if ( m_pData )
        {
            ZeroMemory( m_pData, MH_SIZE );
            m_unSize = newsize;
            return true;
        }
        else
        {
            m_unSize = 0;
            return false;
        }
    }
    bool SetHandleData( BYTE* pszData, int length )
    {
        if ( !pszData )
            return false;
        if ( !SetHandleSize(length) )
            return false;
        ZeroMemory( m_pData, length );
        CopyMemory( m_pData, pszData, length );
    }

```

```

    return true;
}
UINT GetHandleSize() { return m_unSize; }
bool SetHandleSize(UINT newsize)
{
    if ( newsize > MH_SIZE )
        return FALSE;
    m_unSize = newsize;
    return true;
}
BYTE GetHandleChar( UINT ix ) { return m_pData[ix]; }
void SetHandleChar( UINT ix, BYTE ch ) { m_pData[ix] = ch; }
BYTE* data() { return m_pData; }
const BYTE* c_data() { return m_pData; }
protected:
    UINT m_unSize;
    BYTE* m_pData;
};
BOOL Base64_Encode(MacHandle& htext, MacHandle& h64, short linelength);
#ifdef _DEBUG
BOOL Base64_Decode(MacHandle& h64, MacHandle& htext);
#endif
#endif // __BASE64_ENC_H__
// AddDlg.cpp
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#include <vcl.h>
#pragma hdrstop
#include "AddDlg.h"
#include "Main.h"
#include "LockPub.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
#define DATATYPE_DISPLAY_STR "RegEx: %s Format: %s"
void AddItemToLockBox(void)
{
    DWORD dwItemID;
    TForm2* frm = new TForm2(Application);
    if (frm->ShowModal() == mrOk)
    {
        if (!tvAddItemToLockbox(TLockboxDataType(frm->CBDataType->ItemIndex),
            frm->EdData->Text.c_str(), frm->EdName->Text.c_str(), &dwItemID))
            ShowMessage("Unable to add item to lockbox");
    }
}
//-----
__fastcall TForm2::TForm2(TComponent* Owner)
: TForm(Owner)
{

```

```

}
//-----
void __fastcall TForm2::CBDataTypeChange(TObject *Sender)
{
    char buf[128];
    wprintf(buf, DATATYPE_DISPLAY_STR, GetCurRegEx(), GetCurFormat());
    LblRegEx->Caption = AnsiString(buf);
    LblRegEx->Visible = true;
}
//-----
AnsiString __fastcall TForm2::GetCurRegEx(void)
{
    return AnsiString(g_StandardExpressions[CBDataType->ItemIndex]);
}
AnsiString __fastcall TForm2::GetCurFormat(void)
{
    return AnsiString(g_StandardFormats[CBDataType->ItemIndex]);
}
// LBStore.cpp
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#pragma hdrstop
#include <vcl.h>
#include <IniFiles.hpp>
#include "LBStore.h"
//-----
#pragma package(smart_init)
AnsiString GetIniFileName(void)
{
    return ExtractFilePath(ParamStr(0)) + "RExpTest.ini";
}
bool InternalAddItemToLockbox(TLockboxItem* lbi)
{
    TIniFile* inifile = new TIniFile(GetIniFileName());
    TStringList* sl = new TStringList();
    AnsiString SectionName;
    __try
    {
        inifile->ReadSections(sl);
        sl->Sorted = true;
        if (sl->Count == 0)
            SectionName = "1";
        else
            SectionName = IntToStr(StrToInt(sl->Strings[sl->Count - 1]) + 1);
        inifile->WriteInteger(SectionName, "Type", lbi->lbd);
        inifile->WriteString(SectionName, "Name", lbi->szDescription);
        inifile->WriteString(SectionName, "Hash", lbi->szHash);
        if ((lbi->lbd == LBDT_STRING) || (lbi->lbd == LBDT_STRING_CI))
            inifile->WriteString(SectionName, "RegEx", lbi->szRegEx);
    }
}

```

```

__finally
{
    delete sl;
    delete inifile;
}
return true;
}

bool InternalRemoveItemFromLockbox(DWORD dwItemID)
{
    TIniFile* inifile = new TIniFile(GetIniFileName());
    __try
    {
        inifile->EraseSection(AnsiString(dwItemID));
    }
    __finally
    {
        delete inifile;
    }
    return true;
}

bool InternalGetLockboxItems(PLockboxItem pli, DWORD* dwItemCnt)
{
    TIniFile* inifile = new TIniFile(GetIniFileName());
    TStringList* sl = new TStringList();
    int iMax, i;
    AnsiString stritem, SectionName;
    PLockboxItem plbi = pli;
    __try
    {
        inifile->ReadSections(sl);
        if (pli == NULL)
            *dwItemCnt = sl->Count; // if pli is NULL, just return count
        else
        {
            iMax = (sl->Count < (int)*dwItemCnt) ? sl->Count : *dwItemCnt;
            sl->Sorted = true;
            for (i = 0; i < iMax; i++)
            {
                memset(plbi, 0, sizeof(TLockboxItem));
                SectionName = sl->Strings[i];
                plbi->dwItemID = StrToInt(SectionName);
                // read type
                plbi->lbdtype = (TLockboxDataType)inifile->ReadInteger(SectionName, "Type", 0);
                // read name
                stritem = inifile->ReadString(SectionName, "Name", "");
                if (!stritem.IsEmpty())
                    strncpy(plbi->szDescription, stritem.c_str(), sizeof(plbi->szDescription));
                // read hash
                stritem = inifile->ReadString(SectionName, "Hash", "");
                if (!stritem.IsEmpty())

```

```

        strncpy(plbi->szHash, stritem.c_str(), sizeof(plbi->szHash));
// reg regular expression
stritem = infile->ReadString(SectionName, "RegEx", "");
if (!stritem.IsEmpty())
    strncpy(plbi->szRegEx, stritem.c_str(), sizeof(plbi->szRegEx));
    plbi++;
}
}
*dwItemCnt = iMax;
}
}
__finally
{
    delete sl;
    delete infile;
}
return true;
}
// LockPriv.cpp
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#pragma hdrstop
#include <string.h>
#include "LockPriv.h"
#include "rsaapi.h"
#include "base64_Enc.h"
#include "LBStore.h"
//-----
#pragma package(smart_init)
const char* RegExShortcuts[] = {NULL, "[[:alpha:]]", "\\u", "\\l", "\\d", "\\s", "[^[:alpha:]\d\s]"};
void ProcessRegExChar(std::string* str, TRegExChars rec, int& charcnt,
    TRegExChars& lastchar, bool bEnd)
{
    char buf[64];
    if ((!bEnd) && (lastchar == rec))
        charcnt++;
    else if (lastchar != REC_NONE)
    {
        *str += RegExShortcuts[lastchar];
        if (charcnt > 1)
        {
            sprintf(buf, "{%d}", charcnt);
            *str += buf;
        }
        charcnt = 1;
    }
    lastchar = rec;
}
// BuildRegExString() forms a regular expression from a LBDT_STRING* type
// lockbox entry
std::string BuildRegExString(const std::string str, bool bCaseSensitive)

```



```

{
    TRegExChars lastchar = REC_NONE;
    int charcnt = 1;
    std::string strret;
    strret += '(';
    std::string::const_iterator i = str.begin();
    for (; i != str.end(); i++)
    {
        if (isalpha(*i))
        {
            if (bCaseSensitive)
            {
                if (isupper(*i))
                    ProcessRegExChar(&strret, REC_UPPER, charcnt, lastchar, false);
                else if (islower(*i))
                    ProcessRegExChar(&strret, REC_LOWER, charcnt, lastchar, false);
            }
            else
                ProcessRegExChar(&strret, REC_ALPHA, charcnt, lastchar, false);
        }
        else if (isdigit(*i))
            ProcessRegExChar(&strret, REC_DIGIT, charcnt, lastchar, false);
        else if (isspace(*i))
            ProcessRegExChar(&strret, REC_SPACE, charcnt, lastchar, false);
        else
            ProcessRegExChar(&strret, REC_OTHER, charcnt, lastchar, false);
    }
    ProcessRegExChar(&strret, REC_NONE, charcnt, lastchar, true);
    strret += ')';
    return strret;
}

// MD5Hash() returns a base64 encoded MD5 hash of the provided buffer
std::string MD5Hash(unsigned char *buf, unsigned buflen)
{
    MD5_CTX ctx;
    MacHandle mhData, mh64;
    unsigned char hash[16];
    // generate MD5 checksum
    MD5Init(&ctx);
    MD5Update(&ctx, buf, buflen);
    MD5Final(hash, &ctx);
    // base64 encode the checksum
    mhData.SetHandleData(hash, sizeof(hash));
    Base64_Encode(mhData, mh64, 0);
    // and copy to the return buffer
    return std::string((char*)(mh64.c_data()));
}

// FindIt() searches the szStr buffer for the szExpression regular expression,
// and, if found, returns the found data formatted with szFormat
std::string FindIt(const char* szStr, const char* szExpression, const char* szFormat)

```

```

{
    std::string::const_iterator start, end;
    unsigned int flags = boost::match_default;
    std::string result;
    regexp_match_results match;
    std::string search(szStr);
    boost::regex expression(szExpression);
    start = search.begin();
    end = search.end();
    if (regex_search(start, end, match, expression, flags))
        result = regex_format(match, szFormat, boost::format_perl);
    return result;
}

class TPredicate
{
private:
    char* m_szFormat;
    char* m_szHash;
    bool* m_pbMatch;
    bool m_bCase;
public:
    TPredicate(char* szFormat, char* szHash, bool* pbMatch, bool bCase) :
        m_szFormat(szFormat), m_szHash(szHash), m_pbMatch(pbMatch), m_bCase(bCase) {}
    bool operator()(const regexp_match_results& what)
    {
        std::string strhash;
        std::string strmatch = regex_format(what, m_szFormat, boost::format_perl);
        // convert case insensitive data to upper case before hashing
        if (!m_bCase)
            strupr(const_cast<char*>(strmatch.c_str()));
        strhash = MD5Hash((unsigned char*)(strmatch.c_str()), strmatch.length());
        if (strcmp(strhash.c_str(), m_szHash) == 0)
            *m_pbMatch = true; // only set to true, default is false
        return (m_pbMatch); // for now, stop on first find
    }
};

bool FindAll(const char* szStr, const char* szExpression, const char* szFormat,
    const char* szHash, bool bCaseSensitive)
{
    std::string::const_iterator start, end;
    std::string result;
    bool bMatch = false;
    regexp_match_results match;
    std::string search(szStr);
    boost::regex expression(szExpression);
    start = search.begin();
    end = search.end();
    regex_grep(TPredicate(const_cast<char*>(szFormat), const_cast<char*>(szHash),
        &bMatch, bCaseSensitive), start, end, expression);
    return bMatch;
}

```

```

}
bool IsTextInLockbox(char* szStr)
{
    PLockboxItem pli;
    PLockboxItem plitemp;
    int i;
    std::string strex, strfound, strhash;
    DWORD dwItemCnt = 0;
    bool bRet = false;
    if (InternalGetLockboxItems(NULL, &dwItemCnt) && (dwItemCnt > 0))
    {
        pli = (PLockboxItem)(malloc(dwItemCnt * sizeof(TLockboxItem)));
        if (pli)
        {
            __try
            {
                InternalGetLockboxItems(pli, &dwItemCnt);
                plitemp = pli;
                for (i = 0; i < (int)dwItemCnt; i++)
                {
                    if ((plitemp->lbd == LBDT_STRING) || (plitemp->lbd == LBDT_STRING_CI))
                        strex = plitemp->szRegEx;
                    else
                        strex = g_StandardExpressions[plitemp->lbd];
                    bRet = FindAll(szStr, strex.c_str(), g_StandardFormats[plitemp->lbd],
                        plitemp->szHash, plitemp->lbd != LBDT_STRING_CI);
                    if (bRet)
                        break;
                    plitemp++;
                }
            }
            /*
            strfound = FindIt(szStr, strex.c_str(), g_StandardFormats[plitemp->lbd]);
            if (!strfound.empty())
            {
                // convert case insensitive data to upper case before hashing
                if (plitemp->lbd == LBDT_STRING_CI)
                    strupr(const_cast<char*>(strfound.c_str()));
                strhash = MD5Hash((unsigned char*)(strfound.c_str()), strfound.length());
                bRet = strcmp(strhash.c_str(), plitemp->szHash) == 0;
                if (bRet)
                    break;
            }
            plitemp++;
            */
        }
    }
    __finally
    {
        free(pli);
    }
}

```

```

}
return bRet;
}
// LockPub.cpp
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#pragma hdrstop
#include <string.h>
#include "LockPub.h"
#include "LockPriv.h"
#include "LBStore.h"
//-----
#pragma package(smart_init)
// Right now there is only one expression/format pair per data type, but
// there could be more than one per data type, which could result in multiple
// regex/format/hash comparisons per single lockbox search
const char* g_StandardExpressions[] = {
    NULL, // LBDT_STRING
    NULL, // LBDT_STRING_CI
    "(\\d{3})[\\-|\\.|\\s]?\\d{3}[\\-|\\.|\\s]?\\d{4}", // LBDT_USPHONE
    "(\\d{3})[\\-|\\.|\\s]?\\d{2}[\\-|\\.|\\s]?\\d{4}", // LBDT_SSN
    "(\\d{4})[\\-|\\.|\\s]?\\d{4}[\\-|\\.|\\s]?\\d{4}[\\-|\\.|\\s]?\\d{4}", // LBDT_VISAMC
    "(\\d{4})[\\-|\\.|\\s]?\\d{6}[\\-|\\.|\\s]?\\d{5}"; // LBDT_AMEX
const char* g_StandardFormats[] = {"$1", "$1", "$1$2$3", "$1$2$3", "$1$2$3$4", "$1$2$3"};
BOOL WINAPI tvAddItemToLockbox(TLockboxDataType lbd, char* szData,
char* szDescription, DWORD* dwItemID)
{
    char* szNormalData;
    char* szExpression;
    std::string restr, formatstr, hashstr;
    TLockboxItem lbi;
    bool bRet;
    bool bCI = lbd == LBDT_STRING_CI;
    // special handling for plain string or case insensitive string items
    if (bCI || (lbd == LBDT_STRING))
    {
        // copy string
        szNormalData = strdup(szData);
        if (!szNormalData)
            return FALSE;
        // convert case insensitive item to uppercase
        if (bCI)
            strupr(szNormalData);
        // create regular expression from string
        restr = BuildRegExString(szNormalData, !bCI);
        strncpy(lbi.szRegEx, restr.c_str(), sizeof(lbi.szRegEx)); // NOTE: potential truncation
        szExpression = (char*)&lbi.szRegEx[0];
    }
    else
    {

```

```

    szNormalData = szData;
    szExpression = const_cast<char*>(g_StandardExpressions[lbdt]);
}
formatstr = FindIt(szNormalData, szExpression, g_StandardFormats[lbdt]);
if (formatstr.empty()) // means data doesn't match with data type
    return FALSE;
hashstr = MD5Hash((unsigned char*)formatstr.c_str(), formatstr.length());
if (hashstr.length() >= sizeof(lbi.szHash))
    return FALSE;
strcpy(lbi.szHash, hashstr.c_str());
lbi.lbdt = lbdt;
strncpy(lbi.szDescription, szDescription, sizeof(lbi.szDescription)); // NOTE: potential truncation
bRet = InternalAddItemToLockbox(&lbi);
if (bRet && dwItemID)
    *dwItemID = lbi.dwItemID;
return bRet;
}
BOOL WINAPI tvRemoveItemFromLockbox(DWORD dwItemID)
{
    return InternalRemoveItemFromLockbox(dwItemID);
}
BOOL WINAPI tvGetLockboxItems(PLockboxItem pli, DWORD* dwItemCnt)
{
    return InternalGetLockboxItems(pli, dwItemCnt);
}
BOOL WINAPI tvSetLockboxItems(PLockboxItem pli, DWORD dwItemCnt)
{
    // not yet implemented
    return FALSE;
}
// Main.cpp
// Copyright (c) 2003 by Zone Labs Inc. All Rights Reserved.
//-----
#include <vcl.h>
#pragma hdrstop
#include "Main.h"
#include "AddDlg.h"
#include <string>
#include "LockPriv.h"
#include "LockPub.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
const char* g_LBTypeNames[] = {"Case sensitive text", "Case insensitive text",
    "Phone number", "Social Security number", "Visa/Mastercard", "American Express"};
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{

```

```

    RefreshLockboxView();
}
//-----
void __fastcall TForm1::BtnTestClick(TObject *Sender)
{
    if (IsTextInLockbox(MemSample->Lines->Text.c_str()))
        ShowMessage("Data match found!");
    else
        ShowMessage("Data match NOT found!");
}
//-----
void __fastcall TForm1::BtnExitClick(TObject *Sender)
{
    Close();
}
//-----
void __fastcall TForm1::ActRemoveItemUpdate(TObject *Sender)
{
    ActRemoveItem->Enabled = LVLockbox->Selected != NULL;
}
//-----
void __fastcall TForm1::ActAddItemExecute(TObject *Sender)
{
    AddItemToLockBox();
    RefreshLockboxView();
}
//-----
void TForm1::RefreshLockboxView(void)
{
    PLockboxItem pli;
    PLockboxItem plitemp;
    TListItem* item;
    int i;
    DWORD dwItemCnt = 0;
    LVLockbox->Items->Clear();
    LVLockbox->Items->BeginUpdate();
    __try
    {
        if (tvGetLockboxItems(NULL, &dwItemCnt) && (dwItemCnt > 0))
        {
            pli = (PLockboxItem)(malloc(dwItemCnt * sizeof(TLockboxItem)));
            if (pli)
            {
                __try
                {
                    tvGetLockboxItems(pli, &dwItemCnt);
                    plitemp = pli;
                    for (i = 0; i < (int)dwItemCnt; i++)
                    {
                        item = LVLockbox->Items->Add();

```

```

        item->Caption = AnsiString(plitemp->szDescription);
        item->Data = (void*)(plitemp->dwItemID);
        item->SubItems->Add(g_LBTypeNames[plitemp->lbdtt]);
        plitemp++;
    }
}
__finally
{
    free(pli);
}
}
}
__finally
{
    LVLockbox->Items->EndUpdate();
}
}
//-----
void __fastcall TForm1::ActRemoveItemExecute(TObject *Sender)
{
    TListItem* item = LVLockbox->Selected;
    if (item)
    {
        if (MessageDlg("Are you sure you wish to remove the selected lockbox item?",
            mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) == mrYes)
        {
            tvRemoveItemFromLockbox(DWORD(item->Data));
            RefreshLockboxView();
        }
    }
}
//-----
void __fastcall TForm1::ActLoadFileExecute(TObject *Sender)
{
    if (OpenDialog->Execute())
        MemSample->Lines->LoadFromFile(OpenDialog->FileName);
}
//-----
void __fastcall TForm1::LVLockboxKeyUp(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    if (Key == VK_DELETE)
    {
        ActRemoveItem->Execute();
    }
}
//-----
// base64_Enc.cpp
/*

```

Dave Winer, dwiner@well.com, UserLand Software, 4/7/97

I built this project using Symantec C++ 7.0.4 on a Mac 9500.

We needed a handle-based Base 64 encoder/decoder. Looked around the net, found a bunch of code that couldn't easily be adapted to in-memory stuff. Most of them work on files to conserve memory. This is inelegant in scripting environments such as Frontier.

Anyway, so I wrote an encoder/decoder. Docs are being maintained on the web, and updates at:

<http://www.scripting.com/midas/base64/>

If you port this code to another platform please put the result up on a website, and send me a pointer. Also send email if you think this isn't a compatible implementation of Base 64 encoding.

BTW, I made it easy to port -- layering out the handle access routines.

Of course there's a small performance penalty for this, and if you don't like it, change it. Thanks!

```
*/
```

```
/* KKNOTE converted from mac memory objects to machandles....
```

```
*/
```

```
//#include "os/os.h"
```

```
//#include "VSNetLibPCH.h"
```

```
//#pragma hdrstop
```

```
//#include <stdio.h>
```

```
#include <windows.h>
```

```
//#include "zonepch.h"
```

```
#include "base64_enc.h"
```

```
static char encodingTable[64] =
```

```
{
```

```
'A','B','C','D','E','F','G','H',
```

```
'I','J','K','L','M','N','O','P',
```

```
'Q','R','S','T','U','V','W','X',
```

```
'Y','Z','a','b','c','d','e','f',
```

```
'g','h','i','j','k','l','m','n',
```

```
'o','p','q','r','s','t','u','v',
```

```
'w','x','y','z','0','1','2','3',
```

```
'4','5','6','7','8','9','+','/'
```

```
};
```

```
BOOL Base64_Encode(MacHandle& htext, MacHandle& h64, short linelength)
```

```
{
```

```
/*
```

```
encode the handle. some funny stuff about linelength -- it only makes sense to make it a multiple of 4. if it's not a multiple of 4, we make it so (by only checking it every 4 characters.
```

```
further, if it's 0, we don't add any line breaks at all.
```

```
*/
```

```
UINT ixtext;
```

```
UINT lentext;
```

```
UINT origsize;
```

```
int ctextremaining;
```

```
BYTE inbuf [3], outbuf [4];
```

```
short i;
```



```

short charsonline = 0, ctcopy;
ixtext = 0;
lentext = htext.GetHandleSize();
while (true)
{
    ctreminating = lentext - ixtext;
    if (ctremaining <= 0)
        break;
    for (i = 0; i < 3; i++) {
        UINT ix = ixtext + i;
        if (ix < lentext)
            inbuf [i] = htext.GetHandleChar(ix);
        else
            inbuf [i] = 0;
    } /*for*/
    outbuf [0] = (inbuf [0] & 0xFC) >> 2;
    outbuf [1] = ((inbuf [0] & 0x03) << 4) | ((inbuf [1] & 0xF0) >> 4);
    outbuf [2] = ((inbuf [1] & 0x0F) << 2) | ((inbuf [2] & 0xC0) >> 6);
    outbuf [3] = inbuf [2] & 0x3F;
    origsize = h64.GetHandleSize();
    if (!h64.SetHandleSize(origsize + 4))
        return (false);
    ctcopy = 4;
    switch (ctremaining) {
        case 1:
            ctcopy = 2;
            break;
        case 2:
            ctcopy = 3;
            break;
    } /*switch*/
    for (i = 0; i < ctcopy; i++)
        h64.SetHandleChar(origsize + i, encodingTable [outbuf [i]]);
    for (i = ctcopy; i < 4; i++)
        h64.SetHandleChar(origsize + i, '=');
    ixtext += 3;
    charsonline += 4;
    if (linelength > 0) { /*DW 4/8/97 -- 0 means no line breaks*/
        if (charsonline >= linelength) {
            charsonline = 0;
            origsize = h64.GetHandleSize();
            if (!h64.SetHandleSize(origsize + 1))
                return (false);
            h64.SetHandleChar(origsize, '\n');
        }
    }
} /*while*/
return (true);
}

#ifdef _DEBUG

```

```
BOOL Base64_Decode(MacHandle& h64, MacHandle& htext)
```

```
{
    UINT ixtext;
    UINT lentext;
    UINT origsize;
    BYTE ch;
    BYTE inbuf [3], outbuf [4];
    short i, ixinbuf;
    boolean flignore;
    boolean flendtext = false;
    ixtext = 0;
    lentext = h64.GetHandleSize();
    ixinbuf = 0;
    while (true)
    {
        if (ixtext >= lentext)
            break;
        ch = h64.GetHandleChar(ixtext++);
        flignore = false;
        if ((ch >= 'A') && (ch <= 'Z'))
            ch = ch - 'A';
        else if ((ch >= 'a') && (ch <= 'z'))
            ch = ch - 'a' + 26;
        else if ((ch >= '0') && (ch <= '9'))
            ch = ch - '0' + 52;
        else if (ch == '+')
            ch = 62;
        else if (ch == '=') /*no op -- can't ignore this one*/
            flendtext = true;
        else if (ch == '/')
            ch = 63;
        else
            flignore = true;
        if (!flignore) {
            short ctcharsinbuf = 3;
            boolean flbreak = false;
            if (flendtext) {
                if (ixinbuf == 0)
                    break;
                if ((ixinbuf == 1) || (ixinbuf == 2))
                    ctcharsinbuf = 1;
                else
                    ctcharsinbuf = 2;
                ixinbuf = 3;
                flbreak = true;
            }
            inbuf [ixinbuf++] = ch;
            if (ixinbuf == 4) {
                ixinbuf = 0;
                outbuf [0] = (inbuf [0] << 2) | ((inbuf [1] & 0x30) >> 4);
```

```

    outbuf [1] = ((inbuf [1] & 0x0F) << 4) | ((inbuf [2] & 0x3C) >> 2);
    outbuf [2] = ((inbuf [2] & 0x03) << 6) | (inbuf [3] & 0x3F);
    origsize = htext.GetHandleSize();
    if (!htext.SetHandleSize(origsize + ctcharsinbuf))
        return (false);
    for (i = 0; i < ctcharsinbuf; i++)
        htext.SetHandleChar(origsize + i, outbuf [i]);
    }
    if (flbreak)
        break;
    }
} /*while*/
return (true);
} /*decodeHandle*/
#endif //ifdef _DEBUG
// RExpTest.cpp
//-----
#include <vcl.h>
#pragma hdrstop
//-----
USEFORM("Main.cpp", Form1);
USEFORM("AddDlg.cpp", Form2);
//-----
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");
        }
        catch (Exception &exception)
        {
            Application->ShowException(&exception);
        }
    }
    return 0;
}
//-----

```

```

// lockbox_dll.h
// Copyright (c) 2003. All Rights Reserved.
// The following ifdef block is the standard way of creating macros which make exporting
// from a DLL simpler. All files within this DLL are compiled with the LOCKBOX_DLL_EXPORTS
// symbol defined on the command line. this symbol should not be defined on any project
// that uses this DLL. This way any other project whose source files include this file see
// LOCKBOX_DLL_API functions as being imported from a DLL, whereas this DLL sees symbols
// defined with this macro as being exported.
#ifdef LOCKBOX_DLL_EXPORTS
#define LOCKBOX_DLL_API __declspec(dllexport)
#else
#define LOCKBOX_DLL_API __declspec(dllimport)
#endif
#pragma pack(push, 4)
struct LockBoxItem
{
    int    index_id;
    int    category_id;
    char   description[128];
    bool   is_encrypted;
    char   regexp[256];
    char   value[128];
    unsigned char hash[21];
    int    length;
};
LOCKBOX_DLL_API int __stdcall LockBoxLoadFile(char *in_path_file_name, char *in_password);
LOCKBOX_DLL_API int __stdcall LockBoxInitStore(char *in_path_file_name);
LOCKBOX_DLL_API int __stdcall LockBoxCloseStore();
LOCKBOX_DLL_API int __stdcall LockBoxSaveStore(char *in_path_file_name, char *in_password);
LOCKBOX_DLL_API int __stdcall LockBoxGetItem(unsigned int in_index, LockBoxItem * out_item);
LOCKBOX_DLL_API int __stdcall LockBoxGetItemsCount(void);
LOCKBOX_DLL_API int __stdcall LockBoxAddItem(LockBoxItem *in_item);
LOCKBOX_DLL_API int __stdcall LockBoxRemoveItem(unsigned int in_index);
LOCKBOX_DLL_API int __stdcall LockBoxUpdateItem(LockBoxItem * in_item);
LOCKBOX_DLL_API int __stdcall LockBoxHashItem(int in_index);
//Description: Checks buffer for getting lockbox items and replaces all found content by in_wiper
LOCKBOX_DLL_API int __stdcall LockBoxFindAndBlockPrivateData(unsigned char *inout_buffer,
    unsigned int in_length, unsigned char in_wiper);
LOCKBOX_DLL_API int __stdcall LockBoxFindAndBlockPrivateData(wchar_t *inout_buffer,
    unsigned int in_length, unsigned char in_wiper);
//Description: Allows transmission (Does not protect privacy information)
LOCKBOX_DLL_API int __stdcall LockBoxDisableProtection();
//Description: Does not allow transmission (protect privacy information)
LOCKBOX_DLL_API int __stdcall LockBoxEnableProtection();
//Description: Registers call back handler for processing obtained data
typedef void (CALLBACK *lockbox_replace_data_callback)(
    unsigned char *inout_data_found_point,
    unsigned int in_data_found_size,
    void *inout_custom_param);
LOCKBOX_DLL_API int __stdcall LockBoxRegisterCallback(

```

```

lockbox_replace_data_callback in_function,
unsigned char *in_buffer,
unsigned int in_buffer_length);
//      unsigned char *in_addr_buffer,
//unsigned int in_buffer_length,
// void* Function);
//processes all registered callbacks
LOCKBOX_DLL_API int __stdcall LockBoxProcessBuffers(
void *inout_custom_callback_param = NULL);
#pragma pack(pop)
// LockBoxCategory.h
// Copyright (c) 2003. All Rights Reserved.
#ifndef LOCKBOXCATEGORY_H_INCLUDED
#define LOCKBOXCATEGORY_H_INCLUDED
#include <string>
#include <list>
struct LockBoxCategoryItem
{
    int category_id;
    std::string regexp;
    bool is_always_encrypted;
};
typedef std::list< LockBoxCategoryItem > lockbox_list_category;
class LockBoxCategory
{
private:
public:
    lockbox_list_category _store;
    LockBoxCategory();
    virtual ~LockBoxCategory();
    int load(std::string in_path_file_name);
    const LockBoxCategoryItem* get_item(int in_category_id);
};
#endif /* LOCKBOXCATEGORY_H_INCLUDED */
// LockBoxLspCore.h
// Copyright (c) 2003. All Rights Reserved.
#ifndef LOCKBOXLSPCORE_H_INCLUDED
#define LOCKBOXLSPCORE_H_INCLUDED
#include "imsA_dll.h"
class IMSA_DLL_API LockBoxLspCore
{
protected:
public:
    LockBoxLspCore();
    virtual ~LockBoxLspCore();
    void filter_content(unsigned char *inout_buffer, int in_buffer_length);
};
#endif /* LOCKBOXLSPCORE_H_INCLUDED */
// LockBoxSingleton.h
// Copyright (c) 2003. All Rights Reserved.

```

```

#ifndef LOCKBOX_SINGLETON_H_INCLUDED
#define LOCKBOX_SINGLETON_H_INCLUDED
#include "util/Singleton.h"
#include "LockBoxLspCore.h"
typedef Singleton<LockBoxLspCore> LockBoxLspSingleton;
#endif /* LOCKBOX_SINGLETON_H_INCLUDED */

// LockBoxStore.h
// Copyright (c) 2003. All Rights Reserved.
#ifndef LOCKPBOXSTORE_H_INCLUDED
#define LOCKPBOXSTORE_H_INCLUDED
#include <string>
#include <list>
#include "lockbox\LockBoxCategory.h"
#define LOCKBOX_HASHSIZE 20
enum {
    LOCKBOX_CAT_CUSTOM, // "???"
    LOCKBOX_CAT_PHONE, // "(\\d{3})(\\.|\\-|\\s)?(\\s)?(\\d{3})(\\.|\\-|\\s)?(\\d{4})"
    LOCKBOX_CAT_SSN, // "(\\d{3})(\\.|\\-|\\s)?(\\d{2})(\\.|\\-|\\s)?(\\d{4})"
    LOCKBOX_CAT_VISA, // "(\\d{4})(\\.|\\-|\\s)?(\\d{4})(\\.|\\-|\\s)?(\\d{4})(\\.|\\-|\\s)?(\\d{4})"
    LOCKBOX_CAT_AMEX, // "(\\d{4})(\\.|\\-|\\s)?(\\d{6})(\\.|\\-|\\s)?(\\d{5})"
    LOCKBOX_CAT_LAST,
};

struct LockBoxPrivatItem
{
    int index_id;
    int category_id;
    std::string description;
    bool is_encrypted;
    unsigned char hash[21];
    std::string regexp;
    std::string value;
    int length;
    bool is_always_encrypted;
};

typedef std::list< LockBoxPrivatItem > lockbox_list_type;
class LockBoxStore
{
private:
    lockbox_list_type _store;
    lockbox_list_type::iterator get_item_as_iterator(int in_index);
public:
    LockBoxCategory _category_store;
    LockBoxStore();
    virtual ~LockBoxStore();
    bool load(std::string in_file_name, std::string in_password);
    bool save(std::string in_file_name, std::string in_password);
    int update_item(LockBoxPrivatItem &in_item);
    bool delete_item(int in_index);
    bool add_item(LockBoxPrivatItem &in_item);
    int get_items_count();

```

```

const LockBoxPrivateItem* get_item(int in_index);
bool encrypt_item(int in_index);
int load_category_store(std::string in_path_file_name);
};
#endif /* LOCKPBOXSTORE_H_INCLUDED */
// LockPrev.h
// Copyright (c) 2003. All Rights Reserved.
#ifndef LOCKPREV_H_INCLUDED
#define LOCKPREV_H_INCLUDED
#include <string>
enum reg_ex_chars { REC_NONE, REC_ALPHA, REC_UPPER, REC_LOWER, REC_DIGIT,
    REC_SPACE, REC_OTHER};
class LockPrev
{
private:
    void process_reg_ex_char(std::string* str, reg_ex_chars rec, int& charcnt,
        reg_ex_chars& lastchar, bool bEnd);
    static unsigned char *memnmem(unsigned char *in_buf,
        const unsigned char *in_pattern,
        unsigned long in_pattern_len,
        unsigned long in_buf_len
    );
public:
    LockPrev();
    virtual ~LockPrev();
    std::string build_reg_ex_string(const std::string str, bool bCaseSensitive);
    std::string find_it(const char* in_content, int in_content_length,
        const char* in_expression, int in_parens_count);
    bool find_item(const char* in_content, int in_content_length,
        std::string in_expression, int in_parens_count, std::string in_hash,
        bool in_case_sensitive);
    bool generate_md5_hash(unsigned char *in_buf, unsigned in_buf_len,
        unsigned char *out_hash);
    bool static find_unencrypted_item_search(unsigned char* in_content,
        int in_content_length, unsigned char *in_searching,
        unsigned int in_searching_length, unsigned char **out_item_in_buffer);
    bool static find_item_using_binary_search(unsigned char* in_content,
        int in_content_length, unsigned char* in_hash, unsigned int in_searching_length,
        unsigned char **out_item_in_buffer);
    std::string static find_item_using_regxep_search(const char* in_content,
        int in_content_length, const char* in_expression,
        unsigned char **out_item_in_buffer, unsigned int &out_item_length);
    bool static generate_sha_hash(unsigned char *in_buf, unsigned in_buf_len,
        unsigned char *out_hash);
};
#endif /* LOCKPREV_H_INCLUDED */
// UpdateStoreStatus.h
// Copyright (c) 2003. All Rights Reserved.
#include <process.h>
unsigned __stdcall UpdateStoreStatusThread( void* pArguments);

```

```

// lockbox_dll.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include "lockbox_dll.h"
#include "lockbox\UpdateStoreStatus.h"
#include "lockbox\LockBoxStore.h"
#include "lockbox\LockPrev.h"
#include "util/Singleton.h"
#include <list>
class RegisteredCallback
{
public:
    RegisteredCallback(lockbox_replace_data_callback in_callback,
        unsigned char *in_buffer, unsigned int in_buffer_len)
        : callback(in_callback), buffer_len(in_buffer_len),
        buffer(in_buffer)
    {
    }
    lockbox_replace_data_callback callback;
    unsigned char    *buffer;
    unsigned int    buffer_len;
    virtual ~RegisteredCallback()
    {
    }
};
typedef std::list<RegisteredCallback> ListOfRegisteredCallbacks;
static ListOfRegisteredCallbacks g_callbacks;
HANDLE thread_handle;
unsigned thread_id;
static bool g_lockbox_disabled = false;
const int min_password_length = 6;
typedef Singleton<LockBoxStore> LockBoxStoreSingleton;
std::auto_ptr<LockBoxStore> LockBoxStoreSingleton::instance(NULL);
BOOL APIENTRY DllMain( HANDLE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
    )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    {
        // thread_handle = (HANDLE)_beginthreadex( NULL, 0,
        // &UpdateStoreStatusThread, NULL, 0, &thread_id );
    }
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
}

```



```

    return TRUE;
}
#define LOCKBOXINI_DIDNOTOPENFILE -2;
// returns values: 0 - OK
// -1 - the password is not valid
// -2 - did not open category file
// -4 - file is empty
LOCKBOX_DLL_API int __stdcall LockBoxInitStore(char *in_path_file_name)
{
    int items_count = LockBoxGetItemsCount();
    for (int index = 0; index < items_count; index++)
        LockBoxRemoveItem(index);
    if(LockBoxGetItemsCount())
        return -1;
    //*****
    // loads file of category description.
    //*****
    return LockBoxStoreSingleton::get_instance()->_category_store.
        load(in_path_file_name);
}
LOCKBOX_DLL_API int __stdcall LockBoxCloseStore()
{
    return 0;
}
LOCKBOX_DLL_API int __stdcall LockBoxSaveStore(char *in_path_file_name, char *in_password)
{
    return LockBoxStoreSingleton::get_instance()->save(in_path_file_name,
        in_password == NULL ? std::string() : std::string(in_password))
        != true;
}
// returns values: 0 - OK
// -1 - could not load file in store
LOCKBOX_DLL_API int __stdcall LockBoxLoadFile(char *in_path_file_name,
    char *in_password)
{
    return LockBoxStoreSingleton::get_instance()->load(in_path_file_name, in_password == NULL
        ? std::string() : std::string(in_password)) != true;
}
// returns value is: 0 - successful, item is found in lockbox
// -1 - failed, is not found it in lockbox store
// -2 - failed, is not valid pointer of out_item;
LOCKBOX_DLL_API int __stdcall LockBoxGetItem(unsigned int in_index, LockBoxItem * out_item)
{
    if(out_item == NULL)
        return -2;
    const LockBoxPrivatItem* private_item = NULL;
    private_item = LockBoxStoreSingleton::get_instance()->get_item(in_index);
    if(private_item == NULL)
        return -1;
    ZeroMemory(out_item, sizeof(LockBoxItem));

```

```

out_item->category_id = private_item->category_id;
strncpy(out_item->description, private_item->description.c_str(),
private_item->description.length());
out_item->index_id = private_item->index_id;
out_item->is_encrypted = private_item->is_encrypted;
strncpy(out_item->regex, private_item->regex.c_str(),
private_item->regex.length());
strncpy(out_item->value, private_item->value.c_str(), private_item->value.
length());
return 0;
}
LOCKBOX_DLL_API int __stdcall LockBoxGetItemsCount(void)
{
return LockBoxStoreSingleton::get_instance()->get_items_count();;
}
// returns value is: 0 - successful, item is found in lockbox
// -1 - failed, is not found it in lockbox store
LOCKBOX_DLL_API int __stdcall LockBoxRemoveItem(unsigned int in_index)
{
if(LockBoxStoreSingleton::get_instance()->delete_item(in_index) == true)
return 0;
return -1;
}
// returns value is: 0 - successful, returns index of added item
// -1 - failed, could not save it in lockbox store
LOCKBOX_DLL_API int __stdcall LockBoxAddItem(LockBoxItem *in_item)
{
LockBoxPrivateItem adding_item;
ZeroMemory(&adding_item, sizeof(LockBoxPrivateItem));
//TODO: must insert checking an ingoing struct to valid values
adding_item.category_id = in_item->category_id;
adding_item.description.assign(in_item->description);
adding_item.is_encrypted = in_item->is_encrypted;
adding_item.regex.assign(in_item->regex);
adding_item.index_id = in_item->index_id;
if(in_item->is_encrypted == false && adding_item.regex.length())
{
unsigned int searched_length;
unsigned char *searched_item = NULL;
adding_item.value = LockPrev::find_item_using_regexp_search(
in_item->value , strlen(in_item->value),
adding_item.regex.c_str(),
&searched_item, searched_length);
}
else
adding_item.value.assign(in_item->value);
if(LockBoxStoreSingleton::get_instance()->add_item(adding_item) == true)
return 0;
return -1;
}

```

```

LOCKBOX_DLL_API int __stdcall LockBoxUpdateItem(LockBoxItem * in_item)
{
    LockBoxPrivatItem updating_item;
    ZeroMemory(&updating_item, sizeof(LockBoxPrivatItem));
    //TODO: must insert checking an ingoing struct to valid values
    updating_item.category_id = in_item->category_id;
    updating_item.description.assign(in_item->description);
    updating_item.is_encrypted = in_item->is_encrypted;
    updating_item.regexp.assign(in_item->regexp);
    updating_item.value.assign(in_item->value);
    updating_item.index_id = in_item->index_id;
    if(LockBoxStoreSingleton::get_instance()->update_item(updating_item) == 0)
        return 0;
    return -1;
}

LOCKBOX_DLL_API int __stdcall LockBoxHashItem(int in_index)
{
    if(LockBoxStoreSingleton::get_instance()->encrypt_item(in_index) == true)
        return 0;
    return -1;
}

static int ReplacePrivateDataWithCallback(
    unsigned char *inout_buffer,
    unsigned int in_length,
    lockbox_replace_data_callback in_callback,
    void *inout_callback_param)
{
    int items_count = LockBoxGetItemsCount();
    for (int index = 0; index < items_count && !g_lockbox_disabled; index++)
    {
        const LockBoxPrivatItem* private_item = NULL;
        private_item = LockBoxStoreSingleton::get_instance()->get_item(index);
        if(private_item == NULL)
            return -1;
        unsigned char *searched_item = NULL;
        if(private_item->is_encrypted == true)//processing encrypted item
        {
            if(private_item->regexp.length() != 0)// Processing item that has regexp
            {
                unsigned char *searched_item = NULL;
                unsigned int searched_length = 0;
                std::string substrings_result;
                unsigned char *buffer_start;
                buffer_start = inout_buffer;
                unsigned int buffer_len = in_length;
                while(true)
                {
                    substrings_result.assign("");
                    substrings_result = LockPrev::find_item_using_regxep_search(
                        (const char*)buffer_start, buffer_len,

```

```

private_item->regex.c_str(),
&searched_item, searched_length);
if(!substrings_result.length())
break;
unsigned char output_hash[21];
int output_hash_length = 20;
LockPrev::generate_sha_hash(
(unsigned char*)substrings_result.c_str(),
substrings_result.length(), output_hash);
if(!memcmp(output_hash, private_item->hash,
output_hash_length))
{
in_callback(searched_item, searched_length,
inout_callback_param);
//memset(searched_item, in_wiper, searched_length);
}
buffer_start = searched_item + searched_length;
buffer_len = in_length - (buffer_start - inout_buffer);
}
}
else//processing STRING category
{
unsigned char *buffer = inout_buffer;
unsigned int buffer_len = in_length;
while(true)
{
if (!LockPrev::find_item_using_binary_search(
buffer, buffer_len,
(unsigned char *)private_item->hash,
private_item->length, &searched_item))
break;
in_callback(searched_item, private_item->length,
inout_callback_param);
buffer = searched_item + private_item->length;
buffer_len = in_length - (buffer - inout_buffer);
//memset(searched_item, in_wiper, );
}
}
}
else//processing unencrypted item
{
if(private_item->regex.length() != 0)// Processing item that has regex
{
unsigned char *searched_item = NULL;
unsigned int searched_length = 0;
std::string substrings_result;
unsigned char *buffer_start;
buffer_start = inout_buffer;
unsigned int buffer_len = in_length;
while(true)

```

```

{
    substrings_result.assign("");
    substrings_result = LockPrev::find_item_using_regxep_search(
        (const char*)buffer_start, buffer_len,
        private_item->regexp.c_str(),
        &searched_item, searched_length);
    if(!substrings_result.length())
        break;
    if (substrings_result == private_item->value)
        in_callback(searched_item, searched_length,
            inout_callback_param);
    //memset(searched_item, in_wiper, searched_length);
    buffer_start = searched_item + searched_length;
    buffer_len = in_length - (buffer_start - inout_buffer);
}
}
else//processing STRING category
{
    unsigned char *buffer = inout_buffer;
    unsigned int buffer_len = in_length;
    while(true)
    {
        if (!LockPrev::find_unencrypted_item_search(
            buffer, buffer_len,
            (unsigned char *)private_item->value.c_str(),
            private_item->value.length(), &searched_item))
            break;
        in_callback(searched_item, private_item
            ->value.length(), inout_callback_param);
        buffer = searched_item + private_item
            ->value.length();
        buffer_len = in_length - (buffer - inout_buffer);
        /*memset(searched_item, in_wiper, private_item
            ->value.length());*/
    }
}
}
}
return 0;
}

static void CALLBACK lockbox_replace_callback(
    unsigned char *inout_data_found_point,
    unsigned int in_data_found_size,
    void *inout_custom_param
)
{
    int replacement = *((int*) inout_custom_param);
    memset(inout_data_found_point, replacement, in_data_found_size);
}

LOCKBOX_DLL_API int __stdcall LockBoxFindAndBlockPrivateData(unsigned char *inout_buffer,

```

```

    unsigned int in_length, unsigned char in_wiper)
{
    int wiper = (unsigned int) in_wiper;
    return ReplacePrivateDataWithCallback(inout_buffer, in_length,
        lockbox_replace_callback, (void*) &wiper);
}
LOCKBOX_DLL_API int __stdcall LockBoxDisableProtection()
{
    g_lockbox_disabled = true;
    return 0;
}
LOCKBOX_DLL_API int __stdcall LockBoxEnableProtection()
{
    g_lockbox_disabled = false;
    return 0;
}
LOCKBOX_DLL_API int __stdcall LockBoxRegisterCallback(
    lockbox_replace_data_callback in_function,
    unsigned char *in_buffer,
    unsigned int in_buffer_length)
{
    g_callbacks.push_back(RegisteredCallback(in_function, in_buffer,
        in_buffer_length));
    return 0;
}
LOCKBOX_DLL_API int __stdcall LockBoxProcessBuffers(
    void *inout_custom_callback_param /*= NULL*/
)
{
    ListOfRegisteredCallbacks::iterator i;
    for (i = g_callbacks.begin(); i != g_callbacks.end(); i++)
    {
        ReplacePrivateDataWithCallback(i->buffer, i->buffer_len,
            i->callback, inout_custom_callback_param);
    }
    return 0;
}
// lockBox_dllTest.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include <cppunit/TestCase.h>
#include <cppunit/extensions/HelperMacros.h>
#include <msvc6/testrunner/TestRunner.h>
#include "../build/lockbox_dll/lockbox_dll.h"
#include "util/BinaryVector.h"
static void CALLBACK replace_with_exclamation_point_callback(
    unsigned char *inout_data_found_point,
    unsigned int in_data_found_size,
    void *in_dummy_param)
{

```

```

    memset(inout_data_found_point, '!', in_data_found_size);
}
static void CALLBACK replace_with_asterisk_callback(
    unsigned char *inout_data_found_point,
    unsigned int in_data_found_size,
    void *in_dummy_param)
{
    memset(inout_data_found_point, '*', in_data_found_size);
}
class LockBoxDLLTest : public CppUnit::TestCase
{
    CPPUNIT_TEST_SUITE(LockBoxDLLTest);
    CPPUNIT_TEST(delete_items_test);
    CPPUNIT_TEST(add_items_test);
    CPPUNIT_TEST(update_items_test);
    CPPUNIT_TEST(hash_items_test);
    CPPUNIT_TEST(find_and_block_private_data_test);
    CPPUNIT_TEST(disable_enable_protection_test);
    CPPUNIT_TEST(register_call_back_test);
    CPPUNIT_TEST(find_and_block_private_unencrypted_data_test);
    CPPUNIT_TEST(save_load_storage_test);
    CPPUNIT_TEST(load_category_test);
    CPPUNIT_TEST_SUITE_END();
private:
public:
    void setUp()
    {
    }
    void tearDown()
    {
        int items_count = LockBoxGetItemsCount();
        for (int index = 0; index < items_count; index++)
            CPPUNIT_ASSERT(!LockBoxRemoveItem(index));
        CPPUNIT_ASSERT(!LockBoxGetItemsCount());
    }
    void delete_items_test()
    {
        CPPUNIT_ASSERT(!LockBoxGetItemsCount());
        LockBoxItem item;
        int max_count = 1000;
        for (int index = 0; index < max_count; index++)
        {
            ZeroMemory(&item, sizeof(LockBoxItem));
            item.index_id = index;
            item.category_id = 1;
            strcpy(item.description, "my confidential phone");
            strcpy(item.value, "289-07-84");
            CPPUNIT_ASSERT(!LockBoxAddItem(&item));
        }
        CPPUNIT_ASSERT(LockBoxGetItemsCount() == max_count);
    }

```

```

}
void add_items_test()
{
    LockBoxItem item;
    int max_count = 1000;
    for (int index = 0; index < max_count; index++)
    {
        ZeroMemory(&item, sizeof(LockBoxItem));
        item.index_id = index;
        item.category_id = 1;
        strcpy(item.description, "my confidential phone");
        strcpy(item.value, "289-07-84");
        CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    }
    CPPUNIT_ASSERT(LockBoxGetItemsCount() == max_count);
    LockBoxItem stored_item;
    CPPUNIT_ASSERT(!LockBoxGetItem(2, &stored_item));
    CPPUNIT_ASSERT(stored_item.index_id == 2);
    CPPUNIT_ASSERT(item.category_id == stored_item.category_id);
    CPPUNIT_ASSERT(!strcmp(item.description, stored_item.description));
    CPPUNIT_ASSERT(item.is_encrypted == stored_item.is_encrypted);
    CPPUNIT_ASSERT(!strcmp(item.value, stored_item.value));
}
void update_items_test()
{
    LockBoxItem item;
    int max_count = 1000;
    for (int index = 0; index < max_count; index++)
    {
        ZeroMemory(&item, sizeof(LockBoxItem));
        item.index_id = index;
        item.category_id = 1;
        strcpy(item.description, "my confidential phone");
        strcpy(item.value, "289-07-84");
        CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    }
    for (index = 0; index < max_count; index++)
    {
        CPPUNIT_ASSERT(!LockBoxGetItem(index, &item));
        item.category_id = 2;
        strcpy(item.description, "my confidential mobile");
        strcpy(item.value, "80296-89-07-84");
        CPPUNIT_ASSERT(!LockBoxUpdateItem(&item));
        LockBoxItem updated_item;
        CPPUNIT_ASSERT(!LockBoxGetItem(index, &updated_item));
        CPPUNIT_ASSERT(updated_item.index_id == index);
        CPPUNIT_ASSERT(item.category_id == updated_item.category_id);
        CPPUNIT_ASSERT(!strcmp(item.description, updated_item.description));
        CPPUNIT_ASSERT(item.is_encrypted == updated_item.is_encrypted);
        CPPUNIT_ASSERT(!strcmp(item.value, updated_item.value));
    }
}

```



```

}
}
void hash_items_test()
{
    LockBoxItem item;
    int max_count = 1000;
    for (int index = 0; index < max_count; index++)
    {
        ZeroMemory(&item, sizeof(LockBoxItem));
        item.index_id = index;
        item.category_id = 1;
        strcpy(item.description, "my confidential phone");
        strcpy(item.value, "289-07-84");
        CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    }
    int items_count = LockBoxGetItemsCount();
    for (index = 0; index < items_count; index++)
    {
        CPPUNIT_ASSERT(!LockBoxHashItem(index));
    }
    for (index = 0; index < items_count; index++)
    {
        CPPUNIT_ASSERT(!LockBoxGetItem(index, &item));
        CPPUNIT_ASSERT(item.is_encrypted == true);
    }
}
void find_and_block_private_data_test()
{
    LockBoxItem item;
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 0;
    item.category_id = 3;
    strcpy(item.description, "my confidential word");
    strcpy(item.value, "Hello");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 1;
    item.category_id = 2;
    strcpy(item.description, "my confidential phone");
    strcpy(item.value, "800-555-1212");
    strcpy(item.regex, "(\\d{3})(\\.|\\|\\.|\\|\\s)?[\\s]?(\\d{3})(\\.|\\|\\.|\\|\\s)?(\\d{4})");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    int items_count = LockBoxGetItemsCount();
    for (int index = 0; index < items_count; index++)
    {
        CPPUNIT_ASSERT(!LockBoxHashItem(index));
    }
    unsigned char content[] =
        "asasas800-555-Hello all1212dsdsdsdsdfsfHello 111-222-1212sf8800-555-1212sdfdfd";
    int content_length = strlen((const char*)&content[0]);

```

```

LockBoxFindAndBlockPrivateData(&content[0], content_length, "**");
CPPUNIT_ASSERT(!memcmp(content, "asasas800-555-***** all1212dsdsdsdsddfsf***** 111-222-
1212sf8*****sdfdfd",
    content_length));
}

void find_and_block_private_unencrypted_data_test()
{
    LockBoxItem item;
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 0;
    item.category_id = 3;
    strcpy(item.description, "my confidential word");
    strcpy(item.value, "Hello");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 1;
    item.category_id = 2;
    strcpy(item.description, "my confidential phone");
    strcpy(item.value, "8005551212");
    strcpy(item.regex, "(\\d{3})[\\s]|\\s[\\s]?[\\s]?(\\d{3})[\\s]|\\s[\\s]?(\\d{4})");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    int items_count = LockBoxGetItemsCount();
    unsigned char content[] =
        "asasas800-555-Hello all1212dsdsdsdsddfsfHello 111-222-1212sf8800 555-1212sdfdfd";
    int content_length = strlen((const char*)&content[0]);
    LockBoxFindAndBlockPrivateData(&content[0], content_length, "**");
    CPPUNIT_ASSERT(!memcmp(content, "asasas800-555-***** all1212dsdsdsdsddfsf***** 111-222-
1212sf8*****sdfdfd",
        content_length));
}

void disable_enable_protection_test()
{
    LockBoxItem item;
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 0;
    item.category_id = 3;
    strcpy(item.description, "my confidential word");
    strcpy(item.value, "1234567890");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    LockBoxDisableProtection();
    BinaryVector etalon_buffer("1234567890");
    BinaryVector buffer(etalon_buffer);
    LockBoxFindAndBlockPrivateData(buffer.begin(), buffer.size(), "**");
    CPPUNIT_ASSERT(buffer == etalon_buffer); // no replacement
    LockBoxRegisterCallback(replace_with_exclamation_point_callback,
        buffer.begin(), buffer.size());
    LockBoxProcessBuffers();
    CPPUNIT_ASSERT(buffer == etalon_buffer); // no replacement
    LockBoxEnableProtection();
    LockBoxProcessBuffers();
}

```

```

CPPUNIT_ASSERT(buffer == BinaryVector("!!!!!!!!!!"));
buffer.assign(etalon_buffer);
LockBoxFindAndBlockPrivateData(buffer.begin(), buffer.size(), '*');
CPPUNIT_ASSERT(buffer == BinaryVector("*****"));
}

void register_call_back_test()
{
    const int buffer_size = 32;
    unsigned char buffer[buffer_size];
    strcpy((char*) buffer, "1234567890123456789012345678901");
    LockBoxItem item;
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 0;
    item.category_id = 3;
    strcpy(item.description, "my confidential word");
    strcpy(item.value, "1234567890");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    LockBoxRegisterCallback(replace_with_exclamation_point_callback,
        buffer, buffer_size);
    LockBoxProcessBuffers();
    CPPUNIT_ASSERT(strcmp((const char*) buffer,
        "!!!!!!!!!!!!!!!!!!!!!!!!!!!!1") == 0);
    LockBoxRegisterCallback(replace_with_asterisk_callback,
        buffer, buffer_size);
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 1;
    item.category_id = 3;
    strcpy(item.description, "blablabla");
    strcpy(item.value, "!!!");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    ZeroMemory(&item, sizeof(LockBoxItem));
    item.index_id = 2;
    item.category_id = 3;
    strcpy(item.description, "foobar");
    strcpy(item.value, "1");
    CPPUNIT_ASSERT(!LockBoxAddItem(&item));
    LockBoxProcessBuffers();
    CPPUNIT_ASSERT(strcmp((const char*) buffer,
        "*****!") == 0);
}

void save_load_storage_test()
{
    LockBoxItem item;
    int max_count = 1000;
    for (int index = 0; index < max_count; index++)
    {
        ZeroMemory(&item, sizeof(LockBoxItem));
        item.index_id = index;
        item.category_id = 1;
        strcpy(item.description, "my confidential phone");
    }
}

```

```

strcpy(item.value , "289-000-1123");
CPPUNIT_ASSERT(!LockBoxAddItem(&item));
}
CPPUNIT_ASSERT(LockBoxSaveStore("testfiles\\lockbox.xml", NULL) == 0);
CPPUNIT_ASSERT(LockBoxInitStore("testfiles\\lockbox_category.dat") == 0);
CPPUNIT_ASSERT(LockBoxLoadFile("testfiles\\lockbox.xml", NULL) == 0);
CPPUNIT_ASSERT(LockBoxGetItemsCount() == max_count);
char etalon_regexp[] = "(\\d{3})[\\s]|\\.|\\-|\\s)?[\\s]?(\\d{3})[\\s]|\\.|\\-|\\s)?(\\d{4})";
for (index = 0; index < max_count; index++)
{
    CPPUNIT_ASSERT(!LockBoxGetItem(index, &item));
    CPPUNIT_ASSERT(item.index_id == index);
    CPPUNIT_ASSERT(item.category_id == 1);
    CPPUNIT_ASSERT(!strcmp(item.description , "my confidential phone"));
    CPPUNIT_ASSERT(item.is_encrypted == false);
    CPPUNIT_ASSERT(!strcmp(item.value , "289-000-1123"));
    CPPUNIT_ASSERT(!strcmp(item.regexp, etalon_regexp));
}
}

void load_category_test()
{
    CPPUNIT_ASSERT(LockBoxInitStore("testfiles\\lockbox_category.dat") == 0);
}
};

CPPUNIT_TEST_SUITE_REGISTRATION(LockBoxDLLTest);
// LockBoxCategory.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include "lockbox/LockBoxCategory.h"
LockBoxCategory::LockBoxCategory()
{
}
LockBoxCategory::~LockBoxCategory()
{
}
#define MAX_STRING    512
#define KEY_NAME_HASH    "Hash"
#define KEY_NAME_REGEXP    "RegExp"
#define KEY_NAME_CATEGORY_ID "Category id"
#define CATEGORY_ID_ENABLED    "ALWAYS"
int LockBoxCategory::load(std::string in_path_file_name)
{
    HANDLE file_handle;
    file_handle = CreateFile(in_path_file_name.c_str(),
        GENERIC_READ,
        FILE_SHARE_READ,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

```

```

if (file_handle == INVALID_HANDLE_VALUE)
    return -2;
DWORD size = ::GetFileSize (file_handle, NULL);
if (size == INVALID_FILE_SIZE)
    return -4;
CloseHandle(file_handle);
char *sections = new char[size];
size = GetPrivateProfileSectionNames(sections, size,
    in_path_file_name.c_str());
char *curr_section_point = sections;
while( strlen(curr_section_point) >0 ){
    LockBoxCategoryItem category_item;
    char ini_value[MAX_STRING];
    ZeroMemory(&category_item, sizeof(LockBoxCategoryItem));
    // get regular expression
    DWORD error_code = GetPrivateProfileString(curr_section_point,
        KEY_NAME_REGEXP, "", ini_value, MAX_STRING,
        in_path_file_name.c_str());
    if(error_code > 0)
        category_item.regexp.assign(ini_value);
    error_code = GetPrivateProfileString(curr_section_point, KEY_NAME_HASH,
        "", ini_value, MAX_STRING, in_path_file_name.c_str());
    if(error_code > 0)
    {
        if(!strcmp(ini_value, CATEGORY_ID_ENABLED))
            category_item.is_always_encrypted = true;
        else
            category_item.is_always_encrypted = false;
    }
    error_code = GetPrivateProfileString(curr_section_point,
        KEY_NAME_CATEGORY_ID, "", ini_value, MAX_STRING, in_path_file_name.c_str());
    if(error_code > 0)
    {
        category_item.category_id = atoi(ini_value);
    }
    _store.push_back(category_item);
    curr_section_point += strlen(curr_section_point) + 1;
}
delete[] sections;
return 0;
}

const LockBoxCategoryItem* LockBoxCategory::get_item(int in_category_id)
{
    lockbox_list_category::const_iterator i;
    for (i = _store.begin(); i != _store.end(); i++)
        if (i->category_id == in_category_id)
            return &(*i);
    return NULL;
}

// lockBoxCategoryTest.cpp

```

```

// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include <cppunit/TestCase.h>
#include <cppunit/extensions/HelperMacros.h>
#include <msvc6/testrunner/TestRunner.h>
#include "lockbox/LockBoxCategory.h"
class LockBoxCategoryTest : public CppUnit::TestCase
{
    CPPUNIT_TEST_SUITE(LockBoxCategoryTest);
    CPPUNIT_TEST(load_category_test);
    CPPUNIT_TEST(get_category_test);
    CPPUNIT_TEST_SUITE_END();
private:
public:
    void setUp()
    {
    }
    void tearDown()
    {
    }
    void load_category_test()
    {
        LockBoxCategory categories;
        CPPUNIT_ASSERT(categories.load("testfiles\\lockbox_category.dat") == 0);
        const int categories_count = 8;
        CPPUNIT_ASSERT(categories._store.size() == categories_count);
        lockbox_list_category::const_iterator i = NULL;
        i = categories._store.begin();
        CPPUNIT_ASSERT(i->category_id == 1);
        char etalon_regexp[] = "(\\d{3})(\\.|\\-|\\s)?[\\s]?(\\d{3})(\\.|\\-|\\s)?(\\d{4})";
        CPPUNIT_ASSERT(!strcmp(i->regexp.c_str(), etalon_regexp));
        CPPUNIT_ASSERT(i->is_always_encrypted == false);
        int loop_count = 0;
        for (i = categories._store.begin(); i != categories._store.end(); i++)
        {
            loop_count ++;
            if(loop_count == categories_count)
            {
                CPPUNIT_ASSERT(i->category_id == 8);
                CPPUNIT_ASSERT(i->regexp.length());
                CPPUNIT_ASSERT(i->is_always_encrypted == true);
            }
        }
        CPPUNIT_ASSERT(loop_count == categories_count);
    }
    void get_category_test()
    {
        LockBoxCategory categories;
        CPPUNIT_ASSERT(categories.load("testfiles\\lockbox_category.dat") == 0);
        const int categories_count = 8;

```

```

CPPUNIT_ASSERT(categories._store.size() == categories_count);
const LockBoxCategoryItem *item;
int seeking_category_id = 1;
item = categories.get_item(seeking_category_id);
CPPUNIT_ASSERT(item->category_id == 1);
char etalon_regexp[] = "(\\d{3})[\\.|\\.|\\.|\\s]?[\\s]?(\\d{3})[\\.|\\.|\\.|\\s]?(\\d{4})";
CPPUNIT_ASSERT(!strcmp(item->regexp.c_str(), etalon_regexp));
CPPUNIT_ASSERT(item->is_always_encrypted == false);
seeking_category_id = 8;
item = categories.get_item(seeking_category_id);
CPPUNIT_ASSERT(item->category_id == 8);
CPPUNIT_ASSERT(item->regexp.length());
CPPUNIT_ASSERT(item->is_always_encrypted == true);
}
};
CPPUNIT_TEST_SUITE_REGISTRATION(LockBoxCategoryTest);
// LockBoxLspCore.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include "lockbox/LockBoxLspCore.h"
#include "../build/lockbox_dll/lockbox_dll.h"
#include <string>
#define CATEGORY_FILE_NAME "c:\\lockbox_test\\category.dat"
#define STORE_FILE_NAME "c:\\lockbox_test\\lockboxstore.xml"
#define STORE_PASSWORD ""
LockBoxLspCore::LockBoxLspCore()
{
    std::string category_file_name;
    std::string path_store_file_name;
    std::string password;
    category_file_name.assign(CATEGORY_FILE_NAME);
    path_store_file_name.assign(STORE_FILE_NAME);
    password.assign(STORE_PASSWORD);
    if(!LockBoxInitStore(category_file_name.begin()))
    {
        LockBoxLoadFile(path_store_file_name.begin(),
            password.begin());
    }
}
LockBoxLspCore::~LockBoxLspCore()
{
    //TODO:must release the store.
}
void LockBoxLspCore::filter_content(unsigned char *inout_buffer,
    int in_buffer_length)
{
    LockBoxFindAndBlockPrivateData(inout_buffer, in_buffer_length, '*');
}
// LockBoxStore.cpp
// Copyright (c) 2003. All Rights Reserved.

```

```

#include "stdafx.h"
#include "lockbox/LockBoxStore.h"
#include "lockbox/LockPrev.h"
#include "util/BinaryVector.h"
#include "util/exceptions/COMException.h"
#include "util/COMInterfaceHolder.h"
#include "util/strings/wide_string.h"
#include <atlbase.h>
const char *LockBox_predefinedRegExp[] = {
    /*LOCKBOX_CAT_CUSTOM,*/ "",
    /*LOCKBOX_CAT_PHONE,*/ "(\\d{3})[\\.|\\.|\\.|\\s]?[\\s]?(\\d{3})[\\.|\\.|\\.|\\s]?(\\d{4})",
    /*LOCKBOX_CAT_SSN,*/ "(\\d{3})[\\.|\\.|\\.|\\s]?(\\d{2})[\\.|\\.|\\.|\\s]?(\\d{4})",
    /*LOCKBOX_CAT_VISA,*/ "(\\d{4})[\\.|\\.|\\.|\\s]?(\\d{4})[\\.|\\.|\\.|\\s]?(\\d{4})[\\.|\\.|\\.|\\s]?(\\d{4})",
    /*LOCKBOX_CAT_AMEX,*/ "(\\d{4})[\\.|\\.|\\.|\\s]?(\\d{6})[\\.|\\.|\\.|\\s]?(\\d{5})",
    /*LOCKBOX_CAT_LAST,*/ "",
};
LockBoxStore::LockBoxStore()
{
    CoInitialize(NULL);
}
LockBoxStore::~~LockBoxStore()
{
    CoUninitialize();
}
class ComStr
{
public:
    CComBSTR str;
    VARIANT variant;
    ComStr( const std::string &s )
        : str(s.c_str())
    {
        variant.vt = VT_BSTR;
        variant.bstrVal = str;
    }
    ~ComStr()
    {
    }
    operator VARIANT() { return variant; }
};
std::string getNodeTextIfAbsent( IXMLDOMNode *node, const char *name, const char *value )
{
    COMInterfaceHolder<IXMLDOMNode> child_node;
    HRESULT hr = node->selectSingleNode(CComBSTR(name),
        child_node.get_interface_ptr());
    if (FAILED(hr))
        throw new COMException("IXMLDOMNodeList::item", hr);
    if (hr == S_FALSE)
        return value;
    CComBSTR bstr;

```



```

child_node->get_text(&bstr.m_str);
wide_string wstr(bstr.m_str, bstr.Length());
std::string str;
wstr.to_utf8(str);
return str;
}

std::string getNodeText( IXMLDOMNode *node, const char *name )
{
    COMInterfaceHolder<IXMLDOMNode> child_node;
    HRESULT hr = node->selectSingleNode(CComBSTR(name),
        child_node.get_interface_ptr());
    if (FAILED(hr))
        throw new COMException("IXMLDOMNodeList::item", hr);
    CComBSTR bstr;
    child_node->get_text(&bstr.m_str);
    wide_string wstr(bstr.m_str, bstr.Length());
    std::string str;
    wstr.to_utf8(str);
    return str;
}

void createXmlDocument( COMInterfaceHolder<IXMLDOMDocument> *xml_dom_document )
{
    HRESULT hr = CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_INPROC_SERVER,
        IID_IXMLDOMDocument, (void**)xml_dom_document->get_interface_ptr());
    if (FAILED(hr) || **xml_dom_document == NULL)
        throw new COMException("IXMLDOMDocument::CoCreateInstance", hr);
    COMInterfaceHolder<IXMLDOMNode> document_node;
    hr = (*xml_dom_document)->QueryInterface(IID_IXMLDOMNode, (void**)
        document_node.get_interface_ptr());
    if (FAILED(hr))
        throw new COMException("IXMLDOMDocument::QueryInterface", hr);
}

bool LockBoxStore::load(std::string in_file_name, std::string in_password)
{
    COMInterfaceHolder<IXMLDOMDocument> xml_dom_document;
    createXmlDocument(&xml_dom_document);
    VARIANT_BOOL is_success;
    HRESULT hr = xml_dom_document->load(ComStr(in_file_name), &is_success);
    if (FAILED(hr))
        throw new COMException("IXMLDOMDocument::load", hr);
    if (!is_success)
        return false;
    COMInterfaceHolder<IXMLDOMNode> settings_node;
    hr = xml_dom_document->selectSingleNode(CComBSTR(
        "lockbox-settings"), settings_node.get_interface_ptr());
    if (FAILED(hr))
        throw new COMException("IXMLDOMNode::selectSingleNode", hr);
    COMInterfaceHolder<IXMLDOMNodeList> props_node_list;
    hr = settings_node->selectNodes(CComBSTR("item"),
        props_node_list.get_interface_ptr());

```

```

if (FAILED(hr))
    throw new COMException("IXMLDOMNode::selectNodes", hr);
long props_node_list_len;
hr = (*props_node_list)->get_length(&props_node_list_len);
if (FAILED(hr))
    throw new COMException("IXMLDOMNodeList::get_length", hr);
_store.clear();
const LockBoxCategoryItem* category_item_point;
for (long i = 0; i < props_node_list_len; i++)
{
    COMInterfaceHolder<IXMLDOMNode> property_node;
    hr = (*props_node_list)->get_item(i, property_node.get_interface_ptr());
    if (FAILED(hr))
        throw new COMException("IXMLDOMNodeList::item", hr);
    LockBoxPrivateItem item;
    item.index_id = i;
    item.category_id = atoi(getNodeText(*property_node, "category_id").c_str());
    item.description = getNodeText(*property_node, "description");
    item.is_encrypted = atoi(getNodeText(*property_node, "is_encrypted").c_str()) != 0;
    BinaryVector bv;
    bv.decode_base64(getNodeText(*property_node, "hash"));
    if (bv.size() > LOCKBOX_HASHSIZE)
        bv.resize(LOCKBOX_HASHSIZE);
    memcpy(item.hash, bv.begin(), bv.size());
    if (item.category_id == LOCKBOX_CAT_CUSTOM)
        item.regexp = getNodeText(*property_node, "regexp");
    else
    {
        if (item.category_id < LOCKBOX_CAT_CUSTOM
            || item.category_id >= LOCKBOX_CAT_LAST)
        {
            throw new COMException("Bounds error", 0);
        }
        category_item_point = _category_store.get_item(item.category_id);
        if(category_item_point == NULL)
            return false;
        item.regexp = category_item_point->regexp;
        item.is_always_encrypted = category_item_point->is_always_encrypted;
    }
    item.value = getNodeTextIfAbsent(*property_node, "value", "");
    item.length = atoi(getNodeTextIfAbsent(*property_node, "length", "0").c_str());
    add_item(item);
}
return true;
}

void addXMLItem( IXMLDOMDocument *doc, IXMLDOMElement *element,
    const char *name, const std::string &value )
{
    COMInterfaceHolder<IXMLDOMNode> xml_node;
    COMInterfaceHolder<IXMLDOMElement> xml_item;

```

```

HRESULT hr = doc->createElement(
    CComBSTR(name), xml_item.get_interface_ptr());
if (FAILED(hr))
    throw new COMException("IXMLDOMDocument::createElement", hr);
xml_item->put_text(CComBSTR(value.c_str()));
hr = element->appendChild(*xml_item, xml_node.get_interface_ptr());
if (FAILED(hr))
    throw new COMException("IXMLDOMDocument::appendChild", hr);
}

bool LockBoxStore::save(std::string in_file_name, std::string in_password)
{
    COMInterfaceHolder<IXMLDOMDocument> xml_dom_document;
    createXmlDocument(&xml_dom_document);
    COMInterfaceHolder<IXMLDOMNode> xml_node;
    COMInterfaceHolder<IXMLDOMElement> xml_element;
    HRESULT hr = xml_dom_document->createElement(
        CComBSTR("lockbox-settings"), xml_element.get_interface_ptr());
    if (FAILED(hr))
        throw new COMException("IXMLDOMDocument::createElement", hr);
    for (lockbox_list_type::iterator i = _store.begin(); i != _store.end(); ++i)
    {
        COMInterfaceHolder<IXMLDOMElement> xml_item;
        HRESULT hr = xml_dom_document->createElement(
            CComBSTR("item"), xml_item.get_interface_ptr());
        if (FAILED(hr))
            throw new COMException("IXMLDOMDocument::createElement", hr);
        char s[32];
        addXMLItem(*xml_dom_document, *xml_item,
            "category_id", itoa(i->category_id, s, 10));
        addXMLItem(*xml_dom_document, *xml_item,
            "description", i->description);
        addXMLItem(*xml_dom_document, *xml_item,
            "is_encrypted", itoa(i->is_encrypted, s, 10));
        BinaryVector bv;
        bv.assign(i->hash, i->hash + LOCKBOX_HASHSIZE);
        addXMLItem(*xml_dom_document, *xml_item, "hash", bv.base64());
        addXMLItem(*xml_dom_document, *xml_item,
            "regexp", i->regexp);
        addXMLItem(*xml_dom_document, *xml_item,
            "value", i->value);
        addXMLItem(*xml_dom_document, *xml_item,
            "length", itoa(i->length, s, 10));
        hr = xml_element->appendChild(*xml_item, xml_node.get_interface_ptr());
        if (FAILED(hr))
            throw new COMException("IXMLDOMDocument::appendChild", hr);
    }
    hr = xml_dom_document->appendChild(*xml_element, xml_node.get_interface_ptr());
    if (FAILED(hr))
        throw new COMException("IXMLDOMDocument::appendChild", hr);
    hr = xml_dom_document->save(ComStr(in_file_name));
}

```

```

if (FAILED(hr))
    throw new COMException("IXMLDOMDocument::save", hr);
return true;
}
const LockBoxPrivatItem* LockBoxStore::get_item(int in_index)
{
    lockbox_list_type::const_iterator i;
    for (i = _store.begin(); i != _store.end(); i++)
        if (i->index_id == in_index)
            return &(*i);
    return NULL;
}
int LockBoxStore::get_items_count()
{
    lockbox_list_type::const_iterator i;
    int record_count = 0;
    for (i = _store.begin(); i != _store.end(); i++)
        record_count ++;
    return record_count;
}
bool LockBoxStore::add_item(LockBoxPrivatItem &in_item)
{
    _store.push_back(in_item);
    return true;
}
bool LockBoxStore::delete_item(int in_index)
{
    lockbox_list_type::iterator i;
    i = get_item_as_iterator(in_index);
    if(i != NULL)
    {
        _store.erase(i);
        return true;
    }
    return false;
}
// returns value: 0 - update is ok
//   -1 - could not find item in storage
//   -2 - could not update(invalid is_secured )
int LockBoxStore::update_item(LockBoxPrivatItem &in_item)
{
    lockbox_list_type::iterator i;
    i = get_item_as_iterator(in_item.index_id);
    if(i != NULL)
    {
        if(i->is_encrypted != in_item.is_encrypted)
        {
            return -2;
        }
    }
    i->category_id = in_item.category_id;

```

```

i->description = in_item.description;
i->regexp = in_item.regexp;
i->value = in_item.value;
i->length = in_item.length;
if(i->is_encrypted == true)
{
    encrypt_item(i->index_id);
}
return 0;
}
return -1;
}
//
bool LockBoxStore::encrypt_item(int in_index)
{
    lockbox_list_type::iterator i;
    i = get_item_as_iterator(in_index);
    if(i != NULL)
    {
        if(LockPrev::generate_sha_hash((unsigned char *)i->value.begin(),
            i->value.length(), i->hash) == true)
        {
            if(!i->regexp.length())
                i->length = i->value.length();
            i->value.erase(i->value.begin(), i->value.end());
            i->is_encrypted = true;
            return true;
        }
    }
    return false;
}
// returns value: NULL - did not find item
//   adr - found item
lockbox_list_type::iterator LockBoxStore::get_item_as_iterator(int in_index)
{
    lockbox_list_type::iterator i;
    for (i = _store.begin(); i != _store.end(); i++)
        if (i->index_id == in_index)
            return i;
    return NULL;
}
int LockBoxStore::load_category_store(std::string in_path_file_name)
{
    return _category_store.load(in_path_file_name);
}
// LockBoxStoreTest.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include <cppunit/TestCase.h>
#include <cppunit/extensions/HelperMacros.h>

```

```

#include <msvc6/testrunner/TestRunner.h>
#include "lockbox/LockBoxStore.h"
#include "util/BinaryVector.h"
class LockBoxStoreTest : public CppUnit::TestCase
{
    CPPUNIT_TEST_SUITE(LockBoxStoreTest);
    CPPUNIT_TEST(navigate_items_test);
    CPPUNIT_TEST(delete_items_test);
    CPPUNIT_TEST(encrypt_item_test);
    CPPUNIT_TEST(testLoad);
    CPPUNIT_TEST(testStore);
    CPPUNIT_TEST(testStoreLoadStore);
    CPPUNIT_TEST_SUITE_END();
private:
    char file_name[260];
    LockBoxStore store;
public:
    void setUp()
    {
        CoInitialize(NULL);
        char temp_path[260];
        temp_path[0] = '\0';
        file_name[0] = '\0';
        GetTempPath(sizeof(temp_path), temp_path);
        GetTempFileName(temp_path, "LBS", 0, file_name);
    }
    void tearDown()
    {
        DeleteFile(file_name);
        CoUninitialize();
    }
    void delete_items_test()
    {
        LockBoxStore _store;
        CPPUNIT_ASSERT(!_store.get_items_count());
        LockBoxPrivateItem item;
        int max_times = 1000;
        for(int index = 0; index < max_times; index++)
        {
            ZeroMemory(&item, sizeof(LockBoxPrivateItem));
            item.category_id = 0;
            item.description.assign("my phone");
            item.is_encrypted = false;
            item.index_id = index;
            item.length = 20;
            item.value.assign("800-555-1212");
            CPPUNIT_ASSERT(_store.add_item(item));
            const LockBoxPrivateItem *stored_item;
            stored_item = _store.get_item(index);
            CPPUNIT_ASSERT(stored_item);
        }
    }
};

```

```

    CPPUNIT_ASSERT(stored_item->description == item.description);
    CPPUNIT_ASSERT(stored_item->category_id == item.category_id);
    CPPUNIT_ASSERT(stored_item->value == item.value);
}
for(index = 0; index < max_times; index++)
{
    _store.delete_item(index);
}
CPPUNIT_ASSERT(!_store.get_items_count());
}
void novigate_items_test()
{
    LockBoxStore _store;
    CPPUNIT_ASSERT(!_store.get_items_count());
    LockBoxPrivatItem item;
    ZeroMemory(&item, sizeof(LockBoxPrivatItem));
    item.category_id = 0;
    item.description.assign("my phone");
    item.is_encrypted = false;
    item.index_id = 1;
    item.length = 20;
    item.value.assign("800-555-1212");
    CPPUNIT_ASSERT(_store.add_item(item));
    CPPUNIT_ASSERT(_store.get_items_count() == 1);
    const LockBoxPrivatItem *stored_item;
    stored_item = _store.get_item(1);
    CPPUNIT_ASSERT(stored_item);
    CPPUNIT_ASSERT(stored_item->description == item.description);
    CPPUNIT_ASSERT(stored_item->category_id == item.category_id);
    CPPUNIT_ASSERT(stored_item->value == item.value);
    LockBoxPrivatItem insert_item;
    ZeroMemory(&insert_item, sizeof(LockBoxPrivatItem));
    insert_item.value.assign("555-666-7777");
    insert_item.index_id = 1;
    CPPUNIT_ASSERT(_store.update_item(insert_item) == 0);
    stored_item = _store.get_item(1);
    CPPUNIT_ASSERT(stored_item);
    CPPUNIT_ASSERT(stored_item->value == insert_item.value);
}
void encrypt_item_test()
{
    LockBoxStore _store;
    CPPUNIT_ASSERT(!_store.get_items_count());
    LockBoxPrivatItem item;
    ZeroMemory(&item, sizeof(LockBoxPrivatItem));
    item.category_id = 0;
    item.description.assign("my phone");
    item.index_id = 1;
    item.length = 20;
    item.value.assign("800-555-1212");

```

```

CPPUNIT_ASSERT(_store.add_item(item));
CPPUNIT_ASSERT(_store.get_items_count() == 1);
const LockBoxPrivatItem *stored_item;
stored_item = _store.get_item(1);
CPPUNIT_ASSERT(stored_item);
CPPUNIT_ASSERT(stored_item->description == item.description);
CPPUNIT_ASSERT(stored_item->category_id == item.category_id);
CPPUNIT_ASSERT(stored_item->value == item.value);
CPPUNIT_ASSERT(_store.encrypt_item(1) == true);
stored_item = _store.get_item(1);
CPPUNIT_ASSERT(stored_item);
CPPUNIT_ASSERT(stored_item->description == item.description);
CPPUNIT_ASSERT(stored_item->category_id == item.category_id);
CPPUNIT_ASSERT(stored_item->is_encrypted == true);
CPPUNIT_ASSERT(stored_item->value.length() == 0);
CPPUNIT_ASSERT(stored_item->regexp.length() == 0);
CPPUNIT_ASSERT(stored_item->hash != NULL);
CPPUNIT_ASSERT(stored_item->length == 12);
}

void storeFile( const char *content )
{
    FILE *fxml = fopen(file_name, "w");
    fwrite(content, 1, strlen(content), fxml);
    fclose(fxml);
}

std::string loadFile()
{
    char content[2000];
    FILE *file = fopen(file_name, "r");
    if (!file)
        return "";
    int size = fread(content, 1, sizeof(content) - 1, file);
    content[size] = '\0';
    fclose(file);
    return content;
}

void testLoad()
{
    storeFile("\
<lockbox-settings>\n\
<item>\n\
<category_id>0</category_id>\n\
<description>Some Description</description>\n\
<is_encrypted>1</is_encrypted>\n\
<hash>MDEAAAAAAAAAAAAAAAAAAAAAAAAA=</hash>\n\
<regexp>^.*?$</regexp>\n\
</item>\n\
<item>\n\
<category_id>1</category_id>\n\
<description>Some Description 2</description>\n\

```



```
<is_encrypted>0</is_encrypted>\n\
<hash>MDEAAAAAAAAAAAAAAAAAAAAAAAAA=</hash>\n\
<value>A Value 2</value>\n\
<length>20</length>\n\
</item>\n\
</lockbox-settings>\n\
");
CPPUNIT_ASSERT(store.load(file_name, ""));
CPPUNIT_ASSERT_EQUAL(2, store.get_items_count());
BinaryVector bv;
bv.resize(20);
bv[0] = 48;
bv[1] = 49;
LockBoxPrivateItem item;
item = *store.get_item(0);
CPPUNIT_ASSERT_EQUAL(0, item.index_id);
CPPUNIT_ASSERT_EQUAL((int)LOCKBOX_CAT_CUSTOM, item.category_id);
CPPUNIT_ASSERT_EQUAL(std::string("Some Description"), item.description);
CPPUNIT_ASSERT(item.is_encrypted);
CPPUNIT_ASSERT(!memcmp(bv.begin(), item.hash, 20));
/*ALTERNATIVE:
memcpy(bv.begin(), item.hash, 20);
CPPUNIT_ASSERT_EQUAL(std::string("MDEAAAAAAAAAAAAAAAAAAAAAAAAA="), bv.base64());*/
CPPUNIT_ASSERT_EQUAL(std::string("^.*$"), item.regexp);
CPPUNIT_ASSERT_EQUAL(std::string(""), item.value);
CPPUNIT_ASSERT_EQUAL(0, item.length);
item = *store.get_item(1);
CPPUNIT_ASSERT_EQUAL(1, item.index_id);
CPPUNIT_ASSERT_EQUAL((int)LOCKBOX_CAT_PHONE, item.category_id);
CPPUNIT_ASSERT_EQUAL(std::string("Some Description 2"), item.description);
CPPUNIT_ASSERT(!item.is_encrypted);
CPPUNIT_ASSERT(!memcmp(bv.begin(), item.hash, 20));
CPPUNIT_ASSERT_EQUAL(std::string("(\\d{3})[\\.|\\-|\\s]?[\\s]?(\\d{3})[\\.|\\-|\\s]?(\\d{4})", item.regexp);
CPPUNIT_ASSERT_EQUAL(std::string("A Value 2"), item.value);
CPPUNIT_ASSERT_EQUAL(20, item.length);
}
void testStore()
{
    LockBoxPrivateItem item;
    item.index_id = 99;
    item.category_id = LOCKBOX_CAT_CUSTOM;
    item.description = "A Description";
    item.is_encrypted = true;
    memset(item.hash, 0, LOCKBOX_HASHSIZE);
    item.hash[0] = '0';
    item.hash[1] = '1';
    item.regexp = "\\d+";
    item.value = "A Value";
    item.length = 10;
    store.add_item(item);
```

```

    CPPUNIT_ASSERT(store.save(file_name, ""));
    CPPUNIT_ASSERT_EQUAL(
        std::string("\
<lockbox-settings>\
<item>\
<category_id>0</category_id>\
<description>A Description</description>\
<is_encrypted>1</is_encrypted>\
<hash>MDEAAAAAAAAAAAAAAAAAAAAA= </hash>\
<regexp>\\d+</regexp>\
<value>A Value</value>\
<length>10</length>\
</item>\
</lockbox-settings>\n\
"),
        std::string(loadFile()));
}

void testStoreLoadStore()
{
    LockBoxPrivateItem item;
    item.index_id = 99;
    item.category_id = LOCKBOX_CAT_CUSTOM;
    item.description = "A Description";
    item.is_encrypted = true;
    memset(item.hash, 0, LOCKBOX_HASHSIZE);
    item.hash[0] = '0';
    item.hash[1] = '1';
    item.regexp = "\\d+";
    item.value = "A Value";
    item.length = 10;
    store.add_item(item);
    CPPUNIT_ASSERT(store.save(file_name, ""));
    std::string content1 = loadFile();
    CPPUNIT_ASSERT(store.load(file_name, ""));
    DeleteFile(file_name);
    CPPUNIT_ASSERT(store.save(file_name, ""));
    CPPUNIT_ASSERT_EQUAL(content1, loadFile());
}

};

CPPUNIT_TEST_SUITE_REGISTRATION(LockBoxStoreTest);

// LockPrev.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include "lockbox/LockPrev.h"
#include "security/base64_Enc.h"
#include <pcre/pcre.h>
#include <memory>
#include "openssl/err.h"
#include "openssl/sha.h"
#include "openssl/md5.h"

```

```

#include "util/BinaryVector.h"
const char* reg_ex_shortcuts[] = { NULL,
    "[a-zA-Z]",
    "[A-Z]",
    "[a-z]",
    "\\d", "\\s",
    "[^a-zA-Z0-9\\s]"};
LockPrev::LockPrev()
{
}
LockPrev::~~LockPrev()
{
}
//*****
// Refactored code
//*****
bool LockPrev::generate_sha_hash(unsigned char *in_buf, unsigned
in_buf_len, unsigned char *out_hash)
{
    SHA_CTX ctx;
    SHA_Init(&ctx);
    SHA_Update(&ctx, in_buf, in_buf_len);
    SHA_Final(out_hash, &ctx);
    return true;
}
// parameters:
// in_content - pointer of content data
// in_content_length - length of content buffer
// in_expression - regexp(using for seaching private data)
// in_parens_count - amount of parents()
std::string LockPrev::find_item_using_regxep_search(const char* in_content,
int in_content_length, const char* in_expression, unsigned char
**out_item_in_buffer, unsigned int &out_item_length)
{
    const char *error_message;
    int error_offset;
    pcre *regexp = pcre_compile(in_expression, NULL, &error_message,
&error_offset, NULL);
    int rc;
    int parens_count = 0;
    rc = pcre_fullinfo(
    regexp,          /* result of pcre_compile() */
    NULL,          /* result of pcre_study(), or NULL */
    PCRE_INFO_CAPTURECOUNT, /* what is required */
    &parens_count); /* where to put the data */
    int number_of_indexes = (parens_count + 1) * 4;
    std::auto_ptr<int> matches_indexes(new int[number_of_indexes]);
    memset(matches_indexes.get(), 0, number_of_indexes * sizeof(int));
    int last_match_pos = 0;
    std::string regexp_result;

```

```

regexp_result.assign("");
int error_code;
if((error_code = pcre_exec(regexp, NULL, in_content,
in_content_length, last_match_pos, 0,
matches_indexes.get(), number_of_indexes) > 0))
{
const int offset_parens_result = 2;
const int regexp_match_start_index = 0;
const int regexp_match_end_index = 1;
for(int index = 0; index < parens_count; index++)
{
int regexp_match_begin = matches_indexes.get()
[offset_parens_result + 2*index + regexp_match_start_index] ;
int regexp_match_end = matches_indexes.get()
[offset_parens_result + 2*index + regexp_match_end_index] ;
regexp_result.append(in_content + regexp_match_begin,
regexp_match_end - regexp_match_begin);
if(!index)
{
int virtual_parent_count = parens_count - 1;
int offset_end_of_string = matches_indexes.get()
[offset_parens_result + 2*virtual_parent_count +
regexp_match_end_index] ;
*out_item_in_buffer = ((unsigned char*)(in_content)) +
regexp_match_begin;
out_item_length = offset_end_of_string - regexp_match_begin;
}
}
}
pcre_free(regexp);
return regexp_result;
}

// parameters:
// in_content - pointer of content data
// in_content_length - length of content buffer
bool LockPrev::find_item_using_binary_search(unsigned char* in_content,
int in_content_length, unsigned char* in_hash, unsigned int
in_searching_length, unsigned char **out_item_in_buffer)
{
unsigned char current_hash[21];
int hash_length = 20;
for(int offset = 0; offset < in_content_length - in_searching_length;
offset++)
{
unsigned char *real_buf= in_content + offset;
generate_sha_hash(in_content + offset, in_searching_length,
&current_hash[0]);
if(!memcmp(current_hash, in_hash, hash_length))
{
*out_item_in_buffer = in_content + offset;

```

```

        return true;
    }
}
return false;
}
unsigned char *LockPrev::memnmem(unsigned char *in_buf,
    const unsigned char *in_pattern,
    unsigned long in_pattern_len,
    unsigned long in_buf_len
)
{
    if (in_buf_len < in_pattern_len)
        return NULL;
    for (unsigned long i = 0; i <= in_buf_len - in_pattern_len; i++)
        if (!memcmp(in_buf + i, in_pattern, in_pattern_len))
            return in_buf + i;
    return NULL;
}
// parameters:
// in_content - pointer of content data
// in_content_length - length of content buffer
bool LockPrev::find_unencrypted_item_search(unsigned char* in_content,
    int in_content_length, unsigned char *in_searching,
    unsigned int in_searching_length, unsigned char **out_item_in_buffer)
{
    *out_item_in_buffer = memnmem(in_content, in_searching,
        in_searching_length, in_content_length);
    return *out_item_in_buffer != NULL;
}
//*****
// needed for refactoring
//*****
// build_reg_ex_string() forms a regular expression from a LBDT_STRING* type
// lockbox entry
std::string LockPrev::build_reg_ex_string(const std::string str,
    bool bCaseSensitive)
{
    reg_ex_chars lastchar = REC_NONE;
    int charcnt = 1;
    std::string strret;
    strret += '(';
    std::string::const_iterator i = str.begin();
    for (; i != str.end(); i++)
    {
        if (isalpha(*i))
        {
            if (bCaseSensitive)
            {
                if (isupper(*i))
                    process_reg_ex_char(&strret, REC_UPPER, charcnt, lastchar, false);

```

```

        else if (islower(*i))
            process_reg_ex_char(&strret, REC_LOWER, charcnt, lastchar, false);
    }
    else
        process_reg_ex_char(&strret, REC_ALPHA, charcnt, lastchar, false);
    }
    else if (isdigit(*i))
        process_reg_ex_char(&strret, REC_DIGIT, charcnt, lastchar, false);
    else if (isspace(*i))
        process_reg_ex_char(&strret, REC_SPACE, charcnt, lastchar, false);
    else
        process_reg_ex_char(&strret, REC_OTHER, charcnt, lastchar, false);
    }
    process_reg_ex_char(&strret, REC_NONE, charcnt, lastchar, true);
    strret += ' ';
    return strret;
}

void LockPrev::process_reg_ex_char(std::string* str, reg_ex_chars rec, int& charcnt,
    reg_ex_chars& lastchar, bool bEnd)
{
    char buf[64];
    if ((!bEnd) && ((lastchar == rec)))
        charcnt++;
    else if (lastchar != REC_NONE)
    {
        *str += reg_ex_shortcuts[lastchar];
        if (charcnt > 1)
        {
            sprintf(buf, "%d", charcnt);
            *str += buf;
        }
        charcnt = 1;
    }
    lastchar = rec;
}

bool LockPrev::find_item(const char* in_content, int in_content_length,
    std::string in_expression, int in_parens_count, std::string in_hash,
    bool in_case_sensitive)
{
    /* std::string regexp_result;
    bool is_found_it = false;
    regexp_result = find_it(in_content, in_content_length, in_expression.c_str(),
        in_parens_count);
    if (!regexp_result.empty())
    {
        // convert case insensitive data to upper case before hashing
        if (in_case_sensitive)
            strupr(const_cast<char*>(regexp_result.c_str()));
        std::string real_hash;
        real_hash = generate_md5_hash((unsigned char*)(regexp_result.c_str()),

```

```

    regexp_result.length());
    if(strcmp(real_hash.c_str(), in_hash.c_str()) == 0)
        is_found_it = true;
    }
    return is_found_it;
*/
    return true;
}

// generate MD5 checksum
bool LockPrev::generate_md5_hash(unsigned char *in_buf, unsigned
    in_buf_len, unsigned char *out_hash)
{
    MD5_CTX ctx;
    MD5_Init(&ctx);
    MD5_Update(&ctx, in_buf, in_buf_len);
    MD5_Final(out_hash, &ctx);
    return true;
}

// LockPrevTest.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#include <cppunit/TestCase.h>
#include <cppunit/extensions/HelperMacros.h>
#include <msvc6/testrunner/TestRunner.h>
#include "lockbox/LockPrev.h"
#include "ini_file.h"
class LockPrevTest : public CppUnit::TestCase
{
    CPPUNIT_TEST_SUITE(LockPrevTest);
    CPPUNIT_TEST(generate_sha_hash_test);
    CPPUNIT_TEST(find_item_using_regxep_search_test);
    CPPUNIT_TEST(find_item_using_binary_search_test);
    CPPUNIT_TEST(find_unencrypted_item_search_test);
    CPPUNIT_TEST(generate_m5_hash_test);
    CPPUNIT_TEST(build_reg_ex_string_test);
    CPPUNIT_TEST(find_item_test);
    CPPUNIT_TEST(find_nodigits_and_symbols_item_test);
    CPPUNIT_TEST(test_speed_sha);
    CPPUNIT_TEST(test_speed_md5);
    CPPUNIT_TEST(bufer_size_speed_test);
    CPPUNIT_TEST_SUITE_END();
private:
    LockPrev lockprev;
public:
    void setUp()
    {
    }
    void tearDown()
    {
    }
}

```

```

// After Refactored
void generate_sha_hash_test()
{
    unsigned char buffer[6]="Hello";
    unsigned char output_hash[21];
    int output_hash_length = 20;
    unsigned char etalon_hash[]={ 0xD7, 0xF5, 0x6F, 0x62, 0xCD, 0xE2, 0xA0,
        0x44, 0xD0, 0x25, 0x9A, 0xDF, 0x01, 0x95, 0x3B, 0xBB, 0x8F, 0x97,
        0x1A, 0x33 };
    LockPrev::generate_sha_hash(buffer, strlen((const char*)&buffer[0]),
        output_hash);
    CPPUNIT_ASSERT(!memcmp(output_hash, etalon_hash,
        output_hash_length));
}

void find_item_using_regxep_search_test()
{
    std::string regexp;
    std::string content;
    std::string output_string;
    output_string.assign("");
    regexp.assign("(\\d{3})[\\.|\\-|\\s]?[\\s]?(\\d{3})[\\.|\\-|\\s]?(\\d{4})");
    content.assign("asasas800-555-1212dsdsdsd");
    unsigned char *searched_item = NULL;
    unsigned int searched_length = 0;
    output_string = LockPrev::find_item_using_regxep_search(content.c_str(),
        content.length(), regexp.c_str(), &searched_item, searched_length);
    CPPUNIT_ASSERT(output_string == "8005551212");
    CPPUNIT_ASSERT(searched_item != NULL);
    CPPUNIT_ASSERT(searched_length == 12);
    CPPUNIT_ASSERT(!memcmp(searched_item, "800-555-1212", 12));
}

void find_item_using_binary_search_test()
{
    std::string content;
    bool is_found;
    unsigned char etalon_hash[]={ 0xD7, 0xF5, 0x6F, 0x62, 0xCD, 0xE2, 0xA0,
        0x44, 0xD0, 0x25, 0x9A, 0xDF, 0x01, 0x95, 0x3B, 0xBB, 0x8F, 0x97,
        0x1A, 0x33 };
    content.assign("asasas800-555-Hello all1212dsdsdsd");
    unsigned char etalon_item[] = "Hello";
    int searching_item_length = 5;
    unsigned char *searched_item = NULL;
    is_found = LockPrev::find_item_using_binary_search(
        (unsigned char *)content.begin(), content.length(), &etalon_hash[0],
        searching_item_length, &searched_item);
    CPPUNIT_ASSERT(searched_item != NULL);
    CPPUNIT_ASSERT(!memcmp(searched_item, etalon_item,
        searching_item_length));
}

void find_unencrypted_item_search_test()

```



```

{
    std::string content;
    bool is_found;
    content.assign("asasas800-555-Hello all1212dsdsdsd");
    unsigned char etalon_item[] = "Hello";
    int searching_item_length = 5;
    unsigned char *searched_item = NULL;
    LockPrev util_class;
    is_found = util_class.find_unencrypted_item_search(
        (unsigned char *)content.begin(), content.length(), &etalon_item[0],
        searching_item_length, &searched_item);
    CPPUNIT_ASSERT(searched_item != NULL);
    CPPUNIT_ASSERT(!memcmp(searched_item, etalon_item,
        searching_item_length));
}
//*****
// Needed refactor
//*****
void build_reg_ex_string_test()
{
    std::string output_regexp;
    std::string str;
    str.assign("Hello");
    output_regexp = lockprev.build_reg_ex_string(str, true);
    CPPUNIT_ASSERT(output_regexp == "[A-Z][a-z]{4}");
}
void generate_md5_hash_test()
{
    unsigned char buffer[5];
    int buffer_size;
    std::string output_hash;
    strcpy((char*)buffer, "Hello");
    buffer_size = sizeof(buffer);
    unsigned char hash[16];
    lockprev.generate_md5_hash(buffer, buffer_size, &hash[0]);
// CPPUNIT_ASSERT(output_hash == "ixqZU8RhEpaoJ6v4xHgE1w==");
}
void find_item_test()
{
/* std::string output_regexp_phone;
std::string test_expression_phone;
std::string output_regexp_string;
std::string test_expression_string;
test_expression_phone.assign("8005551212");
output_regexp_phone = lockprev.build_reg_ex_string(
    test_expression_phone, true);
std::string output_hash_phone = lockprev.generate_md5_hash(
    (unsigned char *)test_expression_phone.c_str(),
    test_expression_phone.length());
CPPUNIT_ASSERT(output_hash_phone == "cipDbNNjHZ/2s7LfP5y50A==");
*/
}

```

```

test_expression_string.assign("Hello CSP folk");
output_regexp_string = lockprev.build_reg_ex_string(
    test_expression_string, true);
CPPUNIT_ASSERT(output_regexp_string ==
    "([A-Z][a-z]{4}\\s[A-Z]{3}\\s[a-z]{4})");
std::string output_hash_string = lockprev.generate_md5_hash(
    (unsigned char *)test_expression_string.c_str(),
    test_expression_string.length());
CPPUNIT_ASSERT(output_hash_string == "ATBUzeBN93Wd7klwGMfhLA==");
FILE * test_file = NULL;
test_file = fopen( "testfiles\\test1.txt", "r+b" );
CPPUNIT_ASSERT(test_file);
char *test_buffer = NULL;
test_buffer = new char[1000001];
memset(test_buffer, 0 , 1000001 * sizeof(char));
int read_count = 0;
read_count = fread( test_buffer, sizeof( char ), 1000000, test_file);
CPPUNIT_ASSERT(read_count == 1000000);
bool is_found_phone = lockprev.find_item(test_buffer, 1000000,
    "(\\d{3})[\\|\\.|\\.|\\|\\s]?[\\s]?(\\d{3})[\\|\\.|\\.|\\|\\s]?(\\d{4})", 3,
    output_hash_phone, false);
bool is_found_string = lockprev.find_item(test_buffer, 1000000,
    output_regexp_string, 1, output_hash_string, false);
fclose(test_file);
delete []test_buffer;
CPPUNIT_ASSERT(is_found_phone == true);
CPPUNIT_ASSERT(is_found_string == true);
*/
}
// Bug: did not find an expression that has no digits and no letters symbols
void find_nodigits_and_symbols_item_test()
{
/* std::string output_regexp;
std::string test_expression;
test_expression.assign("My SS# is 132 13 1324, so there%%###@@@!!!~!@#$$%^&*( ZX");
output_regexp = lockprev.build_reg_ex_string(test_expression, true);
std::string output_hash = lockprev.generate_md5_hash(
    (unsigned char *)test_expression.c_str(),
    test_expression.length());
CPPUNIT_ASSERT(output_hash == "hiLjLOOnql4XbN+RI3Alzg==");
FILE * test_file = NULL;
test_file = fopen( "testfiles\\test1.txt", "r+b" );
CPPUNIT_ASSERT(test_file);
char *test_buffer = NULL;
test_buffer = new char[1000001];
memset(test_buffer, 0 , 1000001 * sizeof(char));
int read_count = 0;
read_count = fread( test_buffer, sizeof( char ), 1000000, test_file);
CPPUNIT_ASSERT(read_count == 1000000);
bool is_found_string = lockprev.find_item(test_buffer, 1000000,

```

```

    output_regexp, 1, output_hash, false);
fclose(test_file);
delete []test_buffer;
CPPUNIT_ASSERT(is_found_string == true);
*/
}
void test_speed_sha()
{
    FILE * output_sha_log;
    output_sha_log = fopen( "testfiles\\sha_log.txt", "a+b" );
    const char stable_body[] = "800-555-1212";
    //int max_times = 1000;
    char convert_buffer[20];
    CIniFile ini;
    ini.SetPath("testfiles\\lockbox_speed.ini");
    CPPUNIT_ASSERT(ini.ReadFile());
    std::string locked_string = ini.GetValue("MAIN", "STRING");
    std::string count = ini.GetValue("MAIN", "COUNT");
    int max_times = atoi(count.c_str());
    clock_t start, finish;
    double duration;
    CPPUNIT_ASSERT(output_sha_log);
    CTime startTime = CTime::GetCurrentTime();
    unsigned char hash[20];
    unsigned char chash[20];
    start = clock();
    for(int i = 0; i < max_times; i++)
    {
        lockprev.generate_sha_hash
        (
            (unsigned char*)(locked_string.begin() + i%2)
            ,
            locked_string.length() - i%2
            ,
            &hash[0]
        );
        chash[i%20] = memcmp(hash,chash,20);
    }
    CTime endTime = CTime::GetCurrentTime();
    finish = clock();
    duration = (double)(finish - start) ;
    fprintf(output_sha_log, "Test Results:\n");
    fprintf(output_sha_log, " Start time : %d:%d:%d\n",
        startTime.GetHour(), startTime.GetMinute(), startTime.GetSecond());
    fprintf(output_sha_log, " End time : %d:%d:%d\n",
        endTime.GetHour(), endTime.GetMinute(), endTime.GetSecond());
    fprintf(output_sha_log, "Duration : %2.2f \n", duration/1000 );
    fprintf(output_sha_log, "String length : %d \n", locked_string.length() );
    fprintf(output_sha_log, "Hashes : %d \n", max_times );
    fprintf(output_sha_log, "Hashes/sec : %2.2f \n", max_times/duration*1000 );

```

```

fclose(output_sha_log);
}
void test_speed_md5()
{
    FILE * output_md5_log;
    output_md5_log = fopen( "testfiles\\md5_log.txt", "a+b" );
    const char stable_body[] ="800-555-1212";
    // int max_times = 1000;
    char convert_buffer[20];
    /*---
    output_md5_log = fopen( "testfiles\\md5_log.txt", "a+t" );
    */
    CIniFile ini;
    ini.SetPath("testfiles\\lockbox_speed.ini");
    CPPUNIT_ASSERT(ini.ReadFile());
    std::string locked_string = ini.GetValue("MAIN", "STRING");
    std::string count = ini.GetValue("MAIN", "COUNT");
    int max_times = atoi(count.c_str());
    clock_t start, finish;
    double duration;
    CPPUNIT_ASSERT(output_md5_log);
    CTime startTime = CTime::GetCurrentTime();
    unsigned char hash[16];
    unsigned char chash[16];
    start = clock();
    for(int i = 0; i < max_times; i++)
    {
        lockprev.generate_md5_hash
        (
            (unsigned char*)(locked_string.begin() + i%2)
            ,
            locked_string.length() - i%2
            ,
            &hash[0]
        );
        chash[i%16] = memcmp(hash,chash,16);
    }
    CTime endTime = CTime::GetCurrentTime();
    finish = clock();
    duration = (double)(finish - start) ;
    fprintf(output_md5_log, "Test Result:\n");
    fprintf(output_md5_log, " Start time : %d:%d:%d\n",
        startTime.GetHour(), startTime.GetMinute(), startTime.GetSecond());
    fprintf(output_md5_log, " End time : %d:%d:%d\n",
        endTime.GetHour(), endTime.GetMinute(), endTime.GetSecond());
    fprintf(output_md5_log, "Duration : %2.2f \n", duration/1000 );
    fprintf(output_md5_log, "String length : %d \n", locked_string.length() );
    fprintf(output_md5_log, "Hashes : %d \n", max_times );
    fprintf(output_md5_log, "Hashes/sec : %2.2f \n", max_times/duration*1000 );
    fclose(output_md5_log);

```

```

}
void fill_random_values_in(char *inout_buffer, int in_buffer_length)
{
    const int max_rand_number = 255;
    srand(max_rand_number);
    for(int index = 0; index < in_buffer_length; index ++)
    {
        inout_buffer[index] = rand();
    }
    inout_buffer[in_buffer_length - 1] = 0;
}
void bufer_size_speed_test()
{
    /* CIniFile ini;
    ini.SetPath("testfiles\\lockbox_speed.ini");
    CPPUNIT_ASSERT(ini.ReadFile());
    int buffer_size = ini.GetValue("MAIN", "BUFFER_SIZE");
    CPPUNIT_ASSERT(buffer_size);
    std::string searching_string = ini.GetValue("MAIN", "STRING");
    CPPUNIT_ASSERT(searching_string.length());
    CPPUNIT_ASSERT(searching_string.length() < buffer_size);
    char *test_buffer = NULL;
    test_buffer = new char[buffer_size];
    fill_random_values_in(test_buffer, buffer_size);
    strcpy(test_buffer + (buffer_size - searching_string.length() - 1),
        searching_string.c_str());
    FILE * result_file_log;
    result_file_log = fopen("testfiles\\lockbox_speed_log.txt", "a+b");
    CPPUNIT_ASSERT(result_file_log);
    std::string output_regexp;
    output_regexp = lockprev.build_reg_ex_string(searching_string, true);
    CPPUNIT_ASSERT(output_regexp.length());
    std::string output_hash = lockprev.generate_md5_hash(
        (unsigned char *)searching_string.c_str(),
        searching_string.length());
    CPPUNIT_ASSERT(output_hash.length());
    clock_t start, finish;
    double duration;
    CTime startTime = CTime::GetCurrentTime();
    start = clock();
    bool is_found_string = lockprev.find_item(test_buffer, buffer_size,
        output_regexp, 1, output_hash, false);
    delete []test_buffer;
    CPPUNIT_ASSERT(is_found_string == true);
    CTime endTime = CTime::GetCurrentTime();
    finish = clock();
    duration = (double)(finish - start);
    fprintf(result_file_log, "Test Result:\n");
    fprintf(result_file_log, " Primary data:\n");
    fprintf(result_file_log, " buffer_size: %lu\n", buffer_size);

```

```

fprintf(result_file_log, " string: %s\n", searching_string.c_str());
fprintf(result_file_log, " regexp: %s\n", output_regexp.c_str());
fprintf(result_file_log, " hash: %s\n", output_hash.c_str());
fprintf(result_file_log, " Start time : %d:%d:%d\n",
    startTime.GetHour(), startTime.GetMinute(), startTime.GetSecond());
fprintf(result_file_log, " End time : %d:%d:%d\n",
    endTime.GetHour(), endTime.GetMinute(), endTime.GetSecond());
fprintf(result_file_log, " Difference : %2.1f\n", duration );
fprintf(result_file_log,
    " A timer tick is approximately equal to 1/CLOCKS_PER_SEC second:\n");
fprintf(result_file_log, " CLOCKS_PER_SEC:%lu\n\n",CLOCKS_PER_SEC);
fclose(result_file_log);
*/
}
};
CPPUNIT_TEST_SUITE_REGISTRATION(LockPrevTest);
// UpdateStoreStatus.cpp
// Copyright (c) 2003. All Rights Reserved.
#include "stdafx.h"
#ifdef UPDATESTORESTATUSTHREAD_H_INCLUDED
#define UPDATESTORESTATUSTHREAD_H_INCLUDED
#include "UpdateStoreStatus.h"
#include <stddef.h>
unsigned __stdcall UpdateStoreStatusThread( void* pArguments )
{
    DWORD wait_status;
    HANDLE change_handle;
    char path[] = "C:\\Test";
    change_handle = FindFirstChangeNotification(path, FALSE,
        FILE_NOTIFY_CHANGE_FILE_NAME);
    if (change_handle == INVALID_HANDLE_VALUE)
        _endthreadex(GetLastError());
    while (TRUE)
    {
        // Wait for notification.
        wait_status = WaitForMultipleObjects(1, &change_handle, FALSE, INFINITE);
        switch (wait_status)
        {
            case WAIT_OBJECT_0:
                // A file was created or deleted in C:\\WINDOWS.
                // Refresh this directory and restart the
                // change notification. RefreshDirectory is an
                // application-defined function.
                // RefreshDirectory(path)
                if ( FindNextChangeNotification(change_handle) == FALSE )
                    _endthreadex(GetLastError());
                break;
            default:
                _endthreadex(GetLastError());
        }
    }
}

```

```
}  
    return 0;  
}  
#endif /* UPDATESTORESTATUSTHREAD_H_INCLUDED */
```